ANÁLISE SINTÁTICA RRP SLR(Ø,1)

Ludmila Canfield Pereira

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS (M.Sc.)

Aprovada por:

Prof. Estevam De Simone
(Presidente)

Prof. José L.M. Rangel Netto

Prof. Michael Stanton

RIO DE JANEIRO, RJ - BRASIL

MAIO DE 1982

PEREIRA, LUDMILA CANFIELD

Análise Sintática RRP SLR(Ø,1) (Rio de Janeiro) 1982.

VII,95p. 29,7c (COPPE-UFRJ, M.Sc., Engenharia de Sistemas e Computação, 1982)

Tese - Univ. Fed. Rio de Janeiro, Fac. de Engenharia.

1.Análise sintática ascendente de gramáticas RRP I.COPPE/UFRJ II.Título(série)

Aos meus pais

AGRADECIMENTOS

Ao Professor Estevam De Simone, pela orientação e estímulo, que permitiu o desenvolvimento desse trabalho.

A Miguel Argollo Júnior e Sérgio Schneider, pela colaboração e incentivos prestados durante esse trabalho.

Ao CNPq e a CAPES pela ajuda financeira, sem a qual seria impossível a realização deste trabalho.

Ao Evaristo e Rielvo pelos desenhos.

A Suely pela datilografia.

Aos meus amigos, colegas, professores e funcionários da COPPE que direta ou indiretamente contribuiram para a execução deste trabalho.

RESUMO

Gramáticas livres de contexto com lados direitos regulares (RRP) têm sido repetidamente apresentadas como a descrição clara, concisa e natural da sintaxe das linguagens de programação. Entretanto, seus analisadores ascendentes deixam a desejar quanto à simplicidade e eficiência. Nesta tese determinamos condições suficientes para que uma gramática pertença à classe RRP SLR(Ø,1). Definimos, ainda, seu analisador sintático, simples e eficiente,e propomos seu gerador com máquina de LERPARAFRENTE usual e máquina de LERPARATRAS que ocupa apenas um bit de espaço em cada ponto da tabela.

ABSTRACT

Context-free regular right part (RRP) grammars has been repeatedly argued as more clear, concise and "natural" way of describing the syntax of programming languages than BNF notation. Their bottom-up parses, however, are not so simple and efficient. We stablish sufficient conditions to decide if a grammar belongs to RRP SLR(Ø,1) class and introduce a new parser for them. Parser's READAHEAD machine is the usual one and it's READBACK machine is just a boolean function.

INDICE

Páginas

1. INTRODUÇÃO

Trataremos neste trabalho de uma classe especial de gramáticas livres de contexto - denominadas GRAMÁTICAS COM LADOS DIREITOS REGULARES, ou abreviadamente RRP (Regular Right Part) |LaLonde,75| - características por apresentarem conjuntos regulares como lados direitos das regras sintáticas (ou produções). Já que expressões regulares e autômatos finitos são descrições finitas convenientes de conjuntos regulares, serão admitidos para a representação de gramáticas RRP.

Para efeito de distinção, quando se fizer necessário, chamaremos uma gramática livre de contexto definida segundo a formulação tradicional de |Chomsky,56/59| de gramática BNF (Backus - Naur Form).

Nos concentraremos no estudo de gramáticas livres de contexto tanto BNF como RRP (esta definida mais adiante) e omitiremos a menção "livre de contexto".

1.1. MOTIVAÇÃO

Uma gramática serve simultaneamente como:

(1) uma descrição formal da estrutura sintática de uma linguagem para fins de programação e

(2) uma especificação formal de um analisador sintático, sendo, por exemplo, entrada de um gerador que o construa algoritmicamente.

É importante termos uma forma de descrição sintática simples, isto é, compacta e clara, e que satisfaça eficien-

temente ambos os requisitos. Propomos a utilização mais frequente de uma forma de descrição sintática mais simples que a utilizada nas gramáticas BNF : as gramáticas RRP.

Para comparação, considere a seguinte linguagem exemplo descrita através de gramáticas BNF e RRP.

a. BNF :

B → begin LD; LST end

LD → LD; d

LD → d

LST → LST; s

LST → s

b. RRP :

B → begin $(d;)^{+}$ $s(;s)^{*}$ end

Parece-nos claro que a concisão da gramática RRP facilita a compreensão. De fato, quem tenta apreender a estrutura sintática de uma linguagem, através de uma gramática BNF, precisa decifrar o significado de subconjuntos de regras relacionadas, de certa forma "reconstruindo" mentalmente as expressões regulares subjacentes.

A notação BNF é desnecessariamente complicada, embora talvez elegante, para descrever simples repetições. Podemos demonstrar isto de forma mais clara, considerando o exemplo de regras equivalentes abaixo:

a. RRP :

LISTA-DE-PARÂMETROS → EXPRESSÃO(,EXPRESSÃO)*

b. BNF :

LISTA-DE-PARÂMETROS → EXPRESSÃO

LISTA-DE-PARÂMETROS → LISTA-DE-PARÂMETROS, EXPRESSÃO

Intuitivamente, percebemos que uma lista de parâmetros nada tem de implicitamente recursivo. Infelizmente, se estamos restritos a gramáticas BNF, somos forçados a usar uma descrição recursiva. Com isso, ao tentarmos compreender a estrutura sintática de uma linguagem, teremos que distinguir entre a recursão que é uma propriedade inerente à linguagem e a recursão introduzida pelo mecanismo de descrição inadequado.

Este esforço adicional é indesejável e desnecessário, e alegamos que a estrutura sintática pode ser descrita de uma forma mais adequada através de gramáticas RRP.

1.2. HISTÓRICO

O uso de expressões regulares em descrições sintáticas é bastante antigo. Em particular, analisadores léxicos são geralmente assim especificados: |DeRemer,74|, |Gries,71| , |Aho & Ullman,72|. A típica definição de identificador, especificada abaixo, é um clássico exemplo:

IDENTIFICADOR → LETRA(LETRA|DÍGITO)*

A maioria das definições de sintaxe em manuais de usuários também as tem utilizado: COBOL|IBM,73|, PL/I|IBM,72|,WFL|BURROUGHS, 74|, PASCAL|WIRTH,75|. A popularidade desta notação corresponde à sua clareza e facilidade de uso.

Desde 1970 vêm sendo feitas propostas teóricas visando um gerador de analisadores sintáticos ascendentes para as gramáticas RRP. O uso de gramáticas RRP para a construção de analisadores sintáticos ascendentes, evitando, inclusive, a duplicidade de descrições para o usuário e o compilador, admite dois caminhos possíveis. |DeRemer,70| esboçou o processo direto de geração do analisador sintático a partir da gramática RRP. Descreveu também |DeRemer,71/75| um gerador que transforma os lados direitos das regras em autômatos finitos determinísticos, reconstituindo então uma gramática BNF, submetida aos métodos de geração usuais. |LaLonde,77| formalizou a teoria necessária ao processo de geração diretamente a partir da gramática RRP, mas não indicou o processo de geração. Em |LaLonde,75| foram fornecidos algoritmos para esse fim, mas sua implementação é impraticável.

A partir do processo generalizado proposto por |LaLonde,75| e das sugestões de |DeRemer,70|, basicamente, definimos uma classe de gramáticas que denominamos gramáticas RRP SLR(Ø,1) e idealizamos seu gerador de analisadores sintáticos. A extensão deste trabalho para gramáticas RRP LR(Ø,K) e derivadas é trivial.

1.3. CONTEÚDO DA TESE

Descrevemos (a) a definição das gramáticas livres de contexto RRP e (b) uma técnica para construir analisadores sintáticos ascendentes ("BOTTOM-UP") para gramáticas RRP SLR (Ø,1).

Veremos, também, que o método proposto de geração de analisadores sintáticos aceita gramáticas das seguintes classes, contidas na classe RRP SLR(Ø,1): gramáticas RRP LR (Ø,Ø), gramáticas BNF SLR(1) e gramáticas BNF LR(Ø).

Em particular, nos dois últimos casos, produz saída equivalente a do gerador usual dessas classes de gramáticas BNF. Podemos dizer, com propriedade, que o gerador proposto contém os geradores BNF correspondentes.

O gerador abrange uma vasta classe de gramáticas, perfeitamente adequada para a descrição das linguagens de programação usuais. As tabelas de controle do analisador sintático são da mesma ordem de grandeza - normalmente menores - que as das gramáticas BNF equivalentes. E o analisador sintático tem complexidade de tempo idêntica a de gramáticas BNF livres de contexto.

1.4. ORGANIZAÇÃO DA TESE

No capítulo 2, introduzimos a notação empregada e a fundamentação teórica necessária. Revemos as noções de expressões regulares, autômatos finitos, e analisadores sintáticos BNF LR(1) e BNF SLR(1).

No capítulo 3, definimos as gramáticas RRP abordando seu conceito estendido de derivação, seus possíveis formatos de representação, e a representação por autômatos finitos dos conjuntos regulares lados direitos de suas regras. Introduzimos também o problema da análise sintática ascendente em

gramáticas RRP e um pequeno histórico de soluções apresentadas anteriormente. E apresentamos o analisador sintático RRP SLR (Ø,1), juntamente com os algoritmos de construção de suas tabelas e a explicação e demonstração do método utilizado. Finalmente, apresentamos conclusões acerca do gerador proposto, discutindo suas vantagens em relação aos similares atuais.

2. FUNDAMENTAÇÃO TEÓRICA

Este capítulo introduz definições básicas de linguagens formais, bem como a notação utilizada. Inclui as noções necessárias de teoria de conjuntos, autômatos finitos, expressões regulares, gramáticas BNF livres de contexto e gramáticas BNF LR(1) e BNF SLR(1).

2.1. CONCEITOS FUNDAMENTAIS

A referência para as noções básicas e notação de teoria de conjuntos e teoria de linguagens é o capítulo 0 de |Aho & Ullman,72|. Destacaremos apenas os conceitos e notações necessários à nossa apresentação.

2.1.1. Definições Básicas e Notação

(a) Um alfabeto é um conjunto de caracteres quaisquer. Será denotado pela letra grega maiúscula Σ .

(b) Uma sequência de caracteres , ou simplesmente sequência , é um conjunto de zero ou mais símbolos dispostos lado a lado. Geralmente as sequências são formadas por caracteres de um alfabeto especificado. A sequência formada de zero símbolos é denominada sequência vazia e é denotada "ε" . Uma sequência genérica será denotada por letras minúsculas finais (t,...,z).

(c) Destacando algumas operações sobre sequências, temos:

1. Se x e y são sequências, então xy é uma sequência denominada concatenação de x e y.

2. O reverso de uma sequência x, denotada por x^R, é a sequência x escrita na ordem inversa.

3. O comprimento de uma sequência x, denotado por $|x|$, é o número de caracteres que compõem x. Isto é, se $x = a_1,\ldots, a_n$, onde a_i é um caracter, o comprimento de x é n.

(d) Vocabulário é um conjunto de palavras (terminais), onde cada palavra é uma sequência de caracteres.

(e) Usaremos a seguinte notação para operações sôbre conjuntos de sequências:

1. Se A e B são conjuntos de sequências então $A \cup B$ é definido como:

$$A \cup B = \{x \mid x \in A \text{ ou } x \in B\}.$$

2. Se A e B são conjuntos de sequências então AB é definido como:

$$AB = \{xy \mid x \in A \text{ e } y \in B\}.$$

3. Se A é um conjunto de sequências então A^+, indicando o fechamento transitivo de A, é definido como:

$$A^+ = A \cup AA \cup AAA \ldots$$

4. Se A é um conjunto de sequências então A* , indicando o fechamento transitivo e reflexivo de A , é definido como:

$$A^* = \{\varepsilon\} \cup A \cup AA \cup AAA...$$

(f) Uma gramática livre de contexto BNF é definida como uma quádrupla da forma $G = (N,\Sigma,P,S)$ onde:

(1) N é um conjunto finito de categorias sintáticas (os não-terminais de G);

(2) Σ é um conjunto finito de palavras (os terminais de G), sendo que N e Σ são conjuntos disjuntos;

(3) P é um conjunto finito de produções (regras de formação de sentenças), onde cada regra é definida como um par de sequências denotado $\alpha \rightarrow \beta$, onde α é uma sequência que contém pelo menos uma categoria sintática;

(4) $S \in N$ é o símbolo inicial da gramática.

Exemplo: $G = (\{A,S\}, \{0,1\}, P, S)$,

$P = \{S \rightarrow 0A1$

$A \rightarrow 00A1$

$A \rightarrow \varepsilon \}$

(g) Convenções:

1. As categorias serão representadas pela letras maiúsculas do início do alfabeto: $N = \{A, B, \ldots S\}$.

2. As palavras serão representadas pelas letras minúsculas do início do alfabeto: $\Sigma = \{a,b,\ldots,s\}$.

3. As sequências de palavras e/ou categorias da gramática serão representadas por letras gregas minúsculas: $(N \cup \Sigma)^* = \{\alpha, \beta, \ldots, \tau\}$.

4. As sequências de palavras serão representadas pelas letras minúsculas do fim do alfabeto: $\Sigma^* = \{t,u,..,z\}$.

(h) Considerando uma regra genérica $A \rightarrow \alpha$, dizemos que A é o lado esquerdo e α é o lado direito da regra.

(i) Uma forma sentencial de uma gramática G é uma sequência de palavras ou categorias.

(j) Uma sentença em G é uma sequência de palavras.

(k) A linguagem gerada por uma gramática G , denotada por L(G) , é o conjunto de sentenças geradas por G.

(ℓ) Seja $G = (N,\Sigma,P,S)$ uma gramática livre de contexto BNF.

(1) Definimos a relação $\Rightarrow_G$ (lida "deriva diretamente") sobre $(N \cup \Sigma)^*$ como segue:
Se $\alpha A \gamma$ é uma sequência em $(N \cup \Sigma)^*$ e $A \rightarrow \delta$ é uma regra em P , então $\alpha A \gamma \Rightarrow_G \alpha \delta \gamma$.

(2) Dizemos que $A \rightarrow \delta$ é uma regra usada na derivação direta de δ a partir de A .

(3) Se $\alpha_1 \Rightarrow_G \alpha_2 \Rightarrow_G \ldots \Rightarrow_G \alpha_n$, $n \geq 1$, dizemos que α_1 <u>deriva</u> α_n ($\alpha_1 \Rightarrow^*_G \alpha_n$) . Em outras palavras, α_1 deriva α_n em "<u>zero ou mais passos</u>" . Portanto,

(a) $\alpha \Rightarrow^*_G \alpha$ para qualquer sequência α , e

(b) Se $\alpha \Rightarrow^*_G \beta$ e $\beta \Rightarrow_G \gamma$, então $\alpha \Rightarrow^*_G \gamma$.

(4) Se $\alpha_1 \Rightarrow_G \alpha_2 \Rightarrow_G \ldots \Rightarrow_G \alpha_n$, $n > 1$, dizemos que α_1 deriva α_n em "<u>um ou mais passos</u>".

(5) Se $\alpha_1 \Rightarrow_G \alpha_2 \Rightarrow_G \ldots \Rightarrow_G \alpha_n$, e a cada passo desta derivação somente a categoria mais à esquerda é substituída, dizemos que esta é uma <u>derivação esquerda</u> ("LEFTMOST"). Analogamente, uma derivação em que a categoria mais à direita é substituída a cada passo, é dita ser uma <u>derivação direita</u> ("RIGHTMOST"). A notação empregada é $\underset{rm}{\Rightarrow}_G$ e $\underset{lm}{\Rightarrow}_G$ para derivações direita e esquerda, respectivamente.

(6) <u>Notação</u>: omitiremos "G" na relação $\Rightarrow_G$ e nas relações derivadas $\Rightarrow^*_G$, $\Rightarrow^+_G$, $\underset{lm}{\Rightarrow}_G$, $\underset{rm}{\Rightarrow}_G$,por questão de simplicidade, sempre que G for conhecida no contexto em questão.

(7) Se $A \Rightarrow \beta$, dizemos que β <u>reduz</u> para A . Uma regra usada para derivar diretamente β a partir de A é também uma regra usada na redução de β para A .

(8) Se $A = \alpha_0 \Rightarrow \alpha_1 \Rightarrow \ldots \Rightarrow \alpha_n = \beta$, dizemos que $\alpha_n, \ldots, \alpha_1, \alpha_0$ é uma sequência de redução de β para A.

(9) Maiores detalhes sobre as definições acima podem ser encontrados no capítulo 2 de |Aho & Ullman,72| e no capítulo 4 de |Aho & Ullman,77|.

2.1.2. CONJUNTOS REGULARES E EXPRESSÕES REGULARES

A referência para esta seção é o capítulo 2 de |Aho & Ullman,72|.

DEFINIÇÃO. Seja Σ um alfabeto finito. Definimos um conjunto regular em Σ, recursivamente, da seguinte maneira:

(1) Φ é um conjunto regular em Σ;

(2) $\{\varepsilon\}$ é um conjunto regular em Σ;

(3) $\{a\}$ é um conjunto regular em Σ, para todo $a \in \Sigma$;

(4) Se P e Q são conjuntos regulares em Σ, então:

(a) $P \cup Q$

(b) PQ

(c) P^*

também o são;

(5) Nada mais é um conjunto regular.

Portanto, um subconjunto de Σ^* é regular se e somente se for Φ, $\{\varepsilon\}$, ou $\{a\}$, para $a \in \Sigma$, ou se puder ser obtido a partir da aplicação de um número finito de operações de união, concatenação e fechamento, sobre estes conjuntos.

Um método sucinto para denotar conjuntos regulares sobre um alfabeto finito Σ são as expressões regulares (ER).

DEFINIÇÃO. Expressões regulares sobre Σ, e os conjuntos regulares que elas denotam (sequências sobre o alfabeto Σ), são definidos recursivamente como segue:

(1) Φ é uma ER que denota o conjunto regular Φ;

(2) ε é uma ER que denota o conjunto regular $\{\varepsilon\}$;

(3) $a \in \Sigma$ é uma ER que denota o conjunto regular $\{a\}$;

(4) Se p e q são ER que denotam os conjuntos regulares P e Q, respectivamente, então:

   (a) $(p|q)$ é uma ER que denota $P \cup Q$;

   (b) (pq) é uma ER que denota PQ;

   (c) $(p)^*$ é uma ER que denota P^*;

(5) Nada mais é uma expressão regular.

Os parênteses redundantes podem ser removidos obedecendo à seguinte ordem de precedência dos operadores: fechamento, concatenação e alternação.

EXEMPLOS: $\Sigma = \{0,1,a,b\}$

(1) 01, denota $\{01\}$

(2) 0^*, denota $\{0\}^*$

(3) $(0|1)^*$, denota $\{0,1\}^*$

(4) $(0|1)^*$ 011, denota o conjunto de todas as sequências formadas por 0 e 1, terminadas por 011

(5) (a|b) (a|b|0|1)* , denota o conjunto de todas as sequências em {0,1,a,b}* que comecem com a ou b .

(6) ((00|11)*(01|10)(00|11)*(01|10)(00|11)*)*, denota o conjunto de todas as sequências de 0 e 1 que contêm um número par de zeros e de '1'(s).

Para cada conjunto regular teremos pelo menos uma expressão regular que o denota. Para cada expressão regular podemos construir o conjunto regular denotado. Infelizmente, para cada conjunto regular há uma infinidade de expressões regulares que o denotam. Duas expressões regulares são <u>equivalentes</u> se denotam o mesmo conjunto.

As expressões regulares têm propriedades algébricas básicas, citadas a seguir. Sejam p, q e r , expressões regulares. Então:

(1) $p|q = q|p$

(2) $\Phi^* = \varepsilon$

(3) $p|(q|r) = (p|q)|r$

(4) $p(qr) = (pq)r$

(5) $p(q|r) = (pq)|(pr)$

(6) $(p|q)r = (pr)|(qr)$

(7) $p\varepsilon = \varepsilon p = p$

(8) $\Phi p = p\,\Phi = \Phi$

(9) $p^* = p|p^*$

(10) $(p^*)^* = p^*$

(11) $p|p = p$

(12) $p|\Phi = p$

A partir daqui não faremos mais distinção entre uma expressão regular e o conjunto denotado. Por exemplo, o símbolo a representará tanto o conjunto {a} como a expressão regular a .

Aceitaremos as seguintes abreviaturas nas expressões regulares, com mesma precedência que o fechamento:

(a) $p^{?} = p|\varepsilon$

(b) $p^{+} = p\ p^{*}$

(c) $p^{£}q = p(qp)^{*}$.

2.2. AUTÔMATOS FINITOS

Esta seção fornece uma fundamentação teórica importante para a compreensão da tese. Introduzimos as definições de autômatos finitos determinísticos e não determinísticos. As referências para esta seção são |Aho & Ullman,72| e |Hopcroft & Ullman,79|.

2.2.1. AUTÔMATO FINITO

Além de gramáticas e expressões regulares, um outro método através do qual podemos especificar de forma finita os conjuntos regulares é o reconhecedor.

O reconhecedor mais simples é denominado de autômato finito (AF) e reconhece conjuntos regulares.

Um AF consiste de um conjunto finito de estados e um conjunto de transições entre os estados que ocorrem de acor-

do com os símbolos de entrada de um alfabeto Σ . Ao contrário das expressões regulares que permitem a geração das sequências de um conjunto regular, o autômato finito permite determinar se uma dada sequência em Σ^* pertence ao conjunto regular.

DEFINIÇÃO. Um autômato finito determinístico (AFD) M sobre um alfabeto Σ é um sistema (K,Σ,δ,q_0,F), onde:

(1) K é um conjunto finito não vazio de estados;

(2) Σ é um alfabeto finito de entrada;

(3) δ é um mapeamento parcial $(K \times \Sigma) \rightarrow K$ chamado função de transição ;

(4) $q_0 \in K$ é o estado inicial ;

(5) $F \subseteq K$ é o conjunto de estados finais, isto é, os estados que indicam aceitação da sequência de entrada.

Para determinar o comportamento futuro de um AFD necessitamos saber apenas:

(1) seu estado corrente, que "resume" a informação concernente aos símbolos de entrada anteriores, e

(2) a sequência de símbolos de entrada a partir do símbolo corrente.

Essas informações fornecem uma descrição instantânea do AFD, que chamaremos de configuração.

DEFINIÇÃO. Se $M = (K,\Sigma,\delta,q_0,F)$ é um AFD, então um par $(q,w) \in (K \times \Sigma^*)$ é uma configuração de M . Uma configuração da forma (q_0,w) é chamada de inicial , e uma da forma (q,ε) onde $q \in F$, é chamada de final.

Inicialmente, o AFD está no estado q_0 e verifica

qual o símbolo mais à esquerda da sequência de entrada. A interpretação de $\delta(q,a) = p$, para $(q, p \in k)$ e $(a \in \Sigma)$, é que estando M no estado q e encontrando o símbolo de entrada a , muda seu estado para p e passa a analisar o próximo símbolo de entrada.

O mapeamento δ foi definido $(k \times \Sigma) \rightarrow k$. Podemos estender seu domínio para $k \times \Sigma^*$ considerando que $\delta(q,ax) = \delta(\delta(q,a),x)$, para cada $x \in \Sigma^*$ e $a \in \Sigma$. Portanto, a interpretação de $\delta(q,x) = p$ é que M , iniciando no estado q e tendo a sequência x na entrada, estará no estado p quando a sequência x tiver sido lida.

Considerando que um movimento de M é representado por uma relação binária '$\vdash$' entre as configurações, dizemos que $C_0 \vdash^k C_k$, $k \geq 1$, se e somente se existirem configurações $C_1,\ldots,C_{k-1}$ tal que $C_i \vdash C_{i+1}$, para todo $0 \leq i < k$. $C \vdash^+ C'$ significa que $C \vdash^k C'$ para algum $k \geq 1$, e $C \vdash^* C'$ significa que $C \vdash^k C'$ para algum $k \geq 0$.

Dizemos que uma sentença x é <u>aceita</u> por M se $\delta(q_0,x) = p$, para $p \in F$. A <u>linguagem definida</u> por M (L(M)) é o conjunto de sequências aceitas por M , isto é,
$\{w \in \Sigma^* \mid (q_0,w) \vdash^* (q,\varepsilon), q \in F\}$.

Frequentemente é conveniente usarmos uma representação gráfica do A.F.D.

<u>DEFINIÇÃO</u>. Seja $M = (K,\Sigma,\delta,q_0,F)$ um AFD.
O <u>grafo de transição</u> , ou <u>diagrama de estados</u> , para M é um dígrafo não ordenado, onde seus nós são rotulados com os nomes dos estados e as arestas (p,q) são rotuladas com a , $(a \in \Sigma)$,

se $\delta(p,a) = q$. O estado inicial é marcado através de uma seta com o rótulo INÍCIO e os estados finais são indicados por um círculo duplo.

EXEMPLO. As especificações de um AFD são as seguintes. $M = (K, \Sigma, \delta, q_0, F)$, onde:

$\Sigma = \{0,1\}$

$K = \{q_0, q_1, q_2, q_3\}$

$F = \{q_0\}$

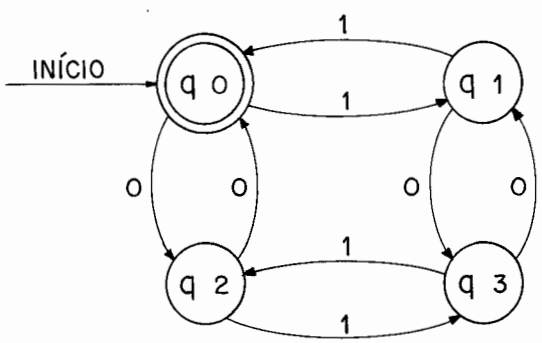
$\delta(q_0,0) = q_2$ $\qquad$ $\delta(q_0,1) = q_1$

$\delta(q_1,0) = q_3$ $\qquad$ $\delta(q_1,1) = q_0$

$\delta(q_2,0) = q_0$ $\qquad$ $\delta(q_2,1) = q_3$

$\delta(q_3,0) = q_1$ $\qquad$ $\delta(q_3,1) = q_2$

O diagrama de estados deste AFD é:



Um AFD pode ter estados equivalentes, sendo portanto um AFD dito não mínimo. Se unirmos os estados equivalentes de um AFD e eliminarmos seus estados inacessíveis, teremos um AFD mínimo , isto é, com o menor número de estados pos-

sível reconhecendo o mesmo conjunto regular. Para todo conjunto regular há um único AFD mínimo que reconhece o conjunto (Teorema de Myhill-Nerode, capítulo 3 de |Hopcroft & Ullman, 79|).

Informalmente, dizemos que dois estados de um AFD são <u>equivalentes</u> se o mesmo conjunto de sequências leva de cada um destes estados a estados finais.

Dizemos que um estado $q \in k$ é <u>inacessível</u> se não existe x tal que:

$$(q_0,x) \vdash^* (q,\varepsilon).$$

Os estados equivalentes são determinados através do particionamento do conjunto de estados acessíveis em classes de equivalência tais que, cada classe contém estados indistinguíveis e é a maior possível. Escolhemos então um estado representativo de cada classe de equivalência para o AFD mínimo.

<u>DEFINIÇÃO</u>. Seja $M = (K,\Sigma,\delta,q_0,F)$ um AFD, e sejam q_1 e q_2 estados distintos. Dizemos que $x \in \Sigma^*$ <u>distingue</u> q_1 de q_2 se $(q_1,x) \vdash^* (q_3,\varepsilon)$, $(q_2,x) \vdash^* (q_4,\varepsilon)$, e apenas um entre q_3 e q_4 é um estado final. Dizemos que q_1 e q_2 são k-indistinguíveis, $q_1 \overset{k}{\equiv} q_2$, se e somente se não existir x , com $|x| \leq k$, que distinga q_1 de q_2 . Dizemos que dois estados q_1 e q_2 são <u>equivalentes</u> , $q_1 \equiv q_2$, se e somente se eles forem k-indistinguíveis para todo $k \geq 0$.

Dizemos que M é <u>mínimo</u> se k não tiver estados inacessíveis ou equivalentes.

O algoritmo de minimização de um AFD proposto por

|Aho & Ullman,72| é a seguinte sequência de passos:

(1) Elimine os estados inacessíveis do AFD;

(2) Construa as relações (classes) de equivalência $\overset{0}{\equiv}$, $\overset{1}{\equiv}$,..., segundo o seguinte método:

(a) Considere q_1 e $q_2 \in K$;

(b) $q_1 \overset{0}{\equiv} q_2$ se e somente se q_1 e $q_2 \in F$ ou q_1 e $q_2 \notin F$;

(c) $q_1 \overset{k}{\equiv} q_2$ se e somente se $q_1 \overset{k-1}{\equiv} q_2$ e para todo $a \in \Sigma$, $\delta(q_1,a) \overset{k-1}{\equiv} \delta(q_2,a)$.

Continue até que $\overset{k+1}{\equiv} = \overset{k}{\equiv}$;

(3) Construa o AFD $M' = (K',\Sigma,\delta',q_0',F')$, onde:

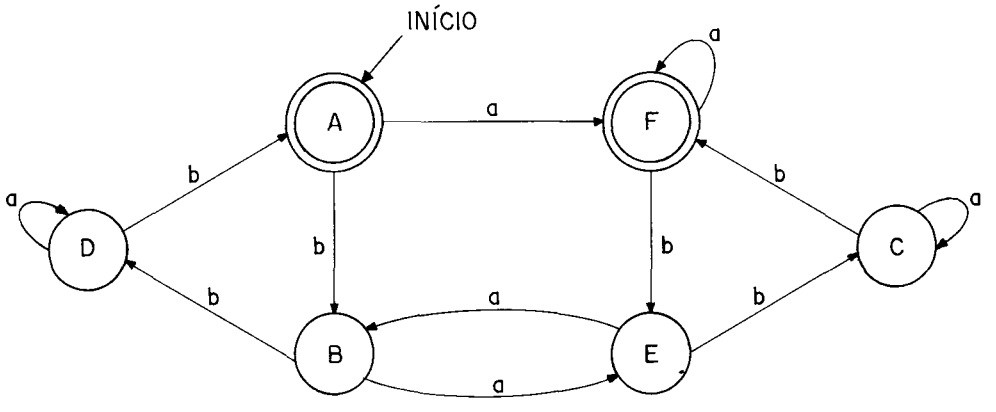
(a) K' é o conjunto de classes de equivalência sob $\equiv$ (isto é, $\overset{k}{\equiv}$). Seja [p] a classe de equivalência do estado p sob $\equiv$;

(b) $\delta'([p],a) = [q]$ se $\delta(p,a) = q$;

(c) q_0' é $[q_0]$;

(d) $F' = \{[q] \mid q \in F\}$.

EXEMPLO. Considere o AFD M não mínimo especificado abaixo:



As classes de equivalência para $\overset{k}{\equiv}$, $k \geq 0$, são as seguintes:

Para $\overset{0}{\equiv}$: {A,F} , {B,C,D,E};

Para $\overset{1}{\equiv}$: {A,F} , {B,E} , {C,D};

Para $\overset{2}{\equiv}$: {A,F} , {B,E} , {C,D}.

Já que $\overset{2}{\equiv} = \overset{1}{\equiv}$, temos que $\equiv = \overset{1}{\equiv}$. O AFD mínimo M' é ({[A], [B], [C]},{a,b}, δ', A, [A]), onde δ' é definido como:

	a	b
[A]	[A]	[B]
[B]	[B]	[C]
[C]	[C]	[A]

Observe que escolhemos [A] para representar a classe de equivalência {A,F} , [B] para representar {B,E} , e [C] para representar {C,D}.

DEFINIÇÃO. Um autômato finito é dito não-determinístico (AFND) se $M = (K,\Sigma,\delta,q_0,F)$ e $\delta : (K \times \Sigma)^* \rightarrow \mathcal{P}(k)$
onde $\mathcal{P}(k)$ é o conjunto dos subconjuntos de k.

Um AFD pode ser considerado um caso particular do AFND. Qualquer conjunto aceito por um AFND também poderá ser aceito por um AFD.

A diferença importante entre o caso determinístico e o não-determinístico é que $\delta(q,a)$ é um (possívelmente vazio) conjunto de estados, ao invés de um único estado. A interpretação de $\delta(q,a) = \{p_1,p_2,\ldots,p_k\}$ é que M , no estado q , tendo "a" como símbolo de entrada, escolhe os estados

$p_1, p_2, \ldots, p_k$ para serem simultaneamente seus próximos estados e lê o próximo simbolo de entrada.

EXEMPLO. Um AFND que aceita o conjunto de todas as sentenças com dois zeros consecutivos ou com dois "1" consecutivos tem a seguinte especificação:

$M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0,1\}, \delta, q_0, \{q_2, q_4\})$

onde:

$\delta(q_0,0) = \{q_0, q_3\}$ $\qquad$ $\delta(q_0,1) = \{q_0, q_1\}$

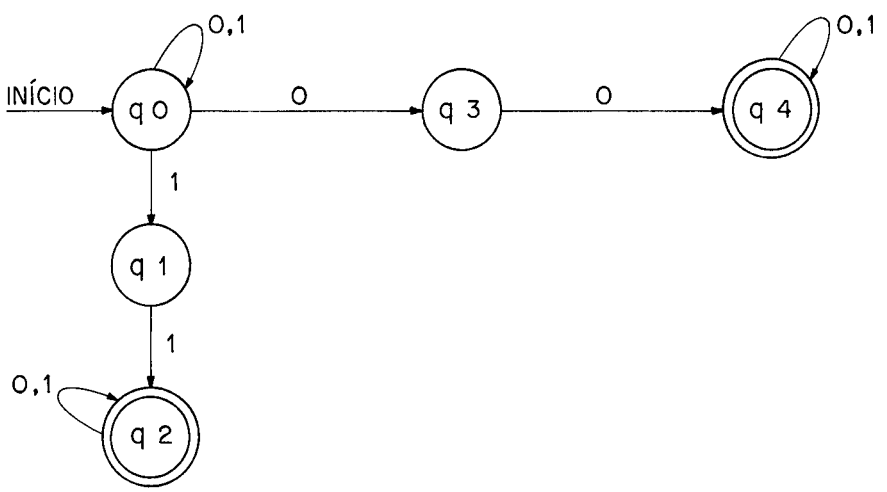
$\delta(q_1,0) = \Phi$ $\qquad$ $\delta(q_1,1) = \{q_2\}$

$\delta(q_2,0) = \{q_2\}$ $\qquad$ $\delta(q_2,1) = \{q_2\}$

$\delta(q_3,0) = \{q_4\}$ $\qquad$ $\delta(q_3,1) = \Phi$

$\delta(q_4,0) = \{q_4\}$ $\qquad$ $\delta(q_4,1) = \{q_4\}$

Seu diagrama de estados é o seguinte:



O AFND fará diversas opções durante a leitura da sequência de entrada. Portanto, suponha que a entrada é 010110. Depois de ler o primeiro zero, M fica nos estados q_0 e q_3. A se-

guir,tendo 1 na entrada, M não pode ir para nenhum estado a partir de q_3, mas a partir de q_0 pode ir para q_0 e q_1. Similarmente, quando o quinto símbolo ("1") for lido, M pode ir de q_1 para q_2 e de q_0 para q_0 e q_1. Portanto, M está nos estados q_0,q_1 e q_2. Já que há uma sequência de estados que leva ao estado final q_2, 01011 é aceito. Da mesma forma, depois da leitura do último símbolo, M está nos estados q_0,q_2 e q_3. Portanto, 010110 é também aceito.

Para todo AFND podemos construir um AFD equivalente, isto é, que aceite a mesma linguagem. O método de construção está inserido no seguinte teorema de|Hopcroft & Ullman,[79]|.

Teorema. Seja L um conjunto aceito por um AFND. Então existe um AFD que aceita L.

Demonstração. Seja $M = (K,\Sigma,\delta,q_0,F)$ um AFND que aceita L. Defina um AFD, $M' = (K',\Sigma,\delta',q_0',F')$ como segue. Os estados de M' são todos os subconjuntos do conjunto de estados de M. Isto é, $K' = \mathcal{P}(k)$. Cada estado do AFD corresponde ao conjunto de estados em que o AFND poderia estar após ler a mesma sequência de entrada lida pelo AFD. F' é o conjunto de todos os estados em K' que contêm um estado final de M. Um elemento de K' será denotado por $[q_1,q_2,\ldots,q_i]$, onde $q_1,q_2,\ldots,q_i$ pertencem a K. Observe que $[q_1,q_2,\ldots,q_i]$ é um único estado do AFD e corresponde a um conjunto de estados do AFND. Note que $q_0' = [q_0]$.

Definimos $\delta'([q_1,q_2,\ldots,q_i],a) = [p_1,p_2,\ldots,p_j]$ se e somente se $\delta(\{q_1,q_2,\ldots,q_i\},a) = \{p_1,p_2,\ldots,p_j\}$. Isto é, δ' aplicado a um elemento $[q_1,q_2,\ldots,q_i]$ de K' é calculado

aplicando-se δ a cada estado de K representado por $[q_1,q_2,\ldots,q_i]$. Aplicando-se δ a cada um dos estados $q_1,q_2,\ldots,q_i$ e fazendo a união, obtemos um novo conjunto de estados $p_1,p_2,\ldots,p_j$. Este novo conjunto de estados tem um representante, $[p_1,p_2,\ldots,p_j]$ em K', e este elemento é o valor de $\delta'([q_1,q_2,\ldots,q_i],a)$.

É fácil demonstrar por indução sobre o comprimento da sequência de entrada x que $\delta'(q_0',x) = [q_1,q_2,\ldots,q_i]$ se e somente se $\delta(q_0,x) = \{q_1,q_2,\ldots,q_i\}$.

Base. O resultado é trivial para $|x| = 0$, já que $q_0' = [q_0]$ e x precisa ser ε.

Indução. Suponha que a hipótese é verdadeira para sequência de entrada de comprimento menor ou igual a m. Seja xa uma sequência de comprimento m+1, com $a \in \Sigma$. Então $\delta'(q_0',xa) = \delta'(\delta'(q_0',x),a)$. Pela hipótese indutiva, $\delta'(q_0',x) = [p_1,p_2,\ldots,p_j]$ se e somente se $\delta(q_0,x) = \{p_1,p_2,\ldots,p_j\}$. Mas, pela definição de δ',

$$\delta'([p_1,p_2,\ldots,p_j],a) = [r_1,r_2,\ldots,r_k]$$

se e somente se

$$\delta(\{p_1,p_2,\ldots,p_j\},a) = \{r_1,r_2,\ldots,r_k\}.$$

Portanto, $\delta'(q_0',xa) = [r_1,r_2,\ldots,r_k]$ se e somente se $\delta(q_0,xa) = \{r_1,r_2,\ldots,r_k\}$, o que dá fundamento à hipótese indutiva.

Finalizando, $\delta'(q_0',x)$ pertence a F' quando $\delta(q_0,x)$ contém um estado de K que pertence a F. Portanto, $L(M) = L(M')$.

EXEMPLO. Seja $M = (\{q_0,q_1\}, \{0,1\}, \delta, q_0, \{q_1\})$ um AFND,onde:

$\delta(q_0,0) = \{q_0,q_1\}$ $\qquad$ $\delta(q_0,1) = \{q_1\}$

$\delta(q_1,0) = \Phi$ $\qquad$ $\delta(q_1,1) = \{q_0,q_1\}$

Podemos contruir um AFD, $M' = (K,\{0,1\}, \delta', [q_0], F)$, que aceite o mesmo conjunto de sentenças aceitas por M , da seguinte forma:

K consiste de todos os subconjuntos de $\{q_0,q_1\}$.

Denotamos os elementos de K por $[q_0]$, $[q_1]$, $[q_0,q_1]$, e Φ.

Já que $\delta(q_0,0) = \{q_0,q_1\}$, $\delta'([q_0],0) = [q_0,q_1]$. Da mesma forma,

$\delta'([q_0],1) = [q_1]$

$\delta'([q_1],0) = \Phi$

$\delta'([q_1],1) = [q_0,q_1]$.

Naturalmente, $\delta'(\Phi,0) = \delta(\Phi,1) = \Phi$.

Finalmente, $\delta'([q_0,q_1], 0) = [q_0,q_1]$, já que $\delta(\{q_0,q_1\},0) = \delta(q_0,0) \cup \delta(q_1,0) = \{q_0,q_1\} \cup \Phi = \{q_0,q_1\}$; e $\delta'([q_0,q_1],1) = [q_0,q_1]$, já que $\delta(\{q_0,q_1\},1) = \delta(q_0,1) \cup \delta(q_1,1) = \{q_1\} \cup \{q_0,q_1\} = \{q_0,q_1\}$.

O conjunto F de estados finais é $\{[q_1],[q_0,q_1]\}$.

Na prática, já que muitos estados do AFND não são acessíveis a partir de $[q_0]$, iniciamos a construção do AFD equivalente com este estado, e só incluímos estados no AFD se eles forem o resultado de uma transição de um estado já incluido.

Assumindo que sempre existe um critério externo pelo qual podemos selecionar um único estado inicial entre os existentes, definiremos abaixo |LaLonde,75| um AF com múltiplas

entradas, que é o tipo que será utilizado na tese.

DEFINIÇÃO. Um AFND com múltiplas entradas é uma 5-tupla $M = (K,\Sigma,\delta,I,F)$ onde:

(a) K é um conjunto finito de estados;

(b) Σ é um conjunto finito de símbolos;

(c) δ , a função de transição, é um mapeamento $(K \times \Sigma \rightarrow K^*)$;

(d) $I \subseteq K$ é o conjunto de estados iniciais;

(e) $F \subseteq K$ é o conjunto de estados finais.

A partir desta definição, introduziremos alguns conceitos e adaptaremos outros já vistos.

Se M contém apenas um estado inicial, então o AFND será dito de "única entrada". Se δ mapeia $(K \times \Sigma \rightarrow K)$, isto é, se cada conjunto no contradomínio da função de transição contém exatamente um membro, M será dito determinístico.

Dizemos que um AFND é trivial se ele reconhece o conjunto vazio.

O processo de transformação AFND múltipla entrada para AFD múltipla entrada, conservando o mesmo número de estados iniciais, é o seguinte |LaLonde,75|:

Seja $M = (K,\Sigma,\delta,I,F)$ um AFND com múltiplas entradas e seja $N = (K',\Sigma,\delta',I',F')$ o seu AFD com múltiplas entradas obtido após a remoção dos estados inúteis de M , onde:

(1) $K' = \mathcal{P}(K) - \{\Phi\}$;

(2) $I' = \{\{p\} \mid p \in I\}$;

(3) $F' = \{S \in K' \mid S \cap F \neq \Phi\}$;

(4) $\delta' = \{((R,a),S) \mid R \in K',\ a \in \Sigma$, e

$S = \{s$ em $\delta(r,a) \mid r \in R\}$ é não vazio$\}$.

Cada estado inicial de N corresponde a um estado inicial de M . Portanto, se o conjunto L é reconhecido por M a partir do estado inicial p , então L é reconhecido por N a partir do estado inicial correspondente {p}.

2.3. ANALISADORES SINTÁTICOS ASCENDENTES

Abordaremos a seguir um tipo de análise sintática (ascendente) de gramáticas livres de contexto BNF, e dois métodos de construção de analisadores sintáticos (LR(1) e SLR(1)). A referência para esta seção é o capítulo 6 de |Aho & Ullman,77|.

DEFINIÇÃO. Uma análise sintática ascendente (bottom-up) de w, de acordo com uma gramática livre de contexto BNF G = (N,Σ,P,S), é a sequência de regras usadas em uma redução canônica de w para S .

EXEMPLO. Seja G uma gramática livre de contexto BNF com as seguintes regras:

A → S

S → aSa

S → bS

S → c

Podemos escrever que $A \overset{*}{\Rightarrow} aabbcaa$ já que:

A => S

=> aSa

=> aaSaa

=> aabSaa

=> aabbSaa

=> aabbcaa

Cada sequência A,S,aSa,aaSaa,etc, é uma forma sentencial de G. A sequência aabbcaa pode ser reduzida a aabbSaa, que por sua vez pode ser reduzida a aabSaa, etc. A sequência de regras usadas na derivação de aabbcaa a partir de A é:

A → S

S → aSa

S → aSa

S → bS

S → bS

S → c

Portanto, uma análise sintática bottom-up de aabbcaa, de acordo com G , é a sequência inversa:

S → c

S → bS

S → bS

S → aSa

S → aSa

A → S

DEFINIÇÃO. Um analisador sintático ascendente para G é uma função que mapeia uma sequência pertencente a L(G) em uma análise sintática ascendente, e uma sequência que não pertença a L(G) em uma mensagem de erro.

Dentro da análise sintática ascendente, um conceito fundamental é o de "handle" (trecho reduzido) de uma forma sen-

tencial.

DEFINIÇÃO. Seja $\alpha = \beta A \delta$ uma forma sentencial de G . Dizemos que x é uma frase da forma sentencial α , para uma categoria A , se $S \overset{*}{\Rightarrow} \beta A \delta$ e $A \overset{+}{\Rightarrow} x$. Dizemos que x é uma frase simples se $S \overset{*}{\Rightarrow} \beta A \delta$ e $A \rightarrow x$. O trecho reduzido de qualquer forma sentencial é a frase simples mais à esquerda.

Para uma grande classe de gramáticas BNF livres de contexto podemos construir analisadores sintáticos ascendentes do tipo LR. São assim chamados porque analisam a entrada da esquerda para a direita (LEFT-TO-RIGHT) e constroem uma derivação direita na ordem inversa.

De uma maneira geral, os analisadores LR apresentam uma série de vantagens sobre os outros tipos de analisadores sintáticos existentes, tais como:

(1) abrangem uma vasta classe de gramáticas, perfeitamente adequada para a descrição das linguagens de programação usuais;

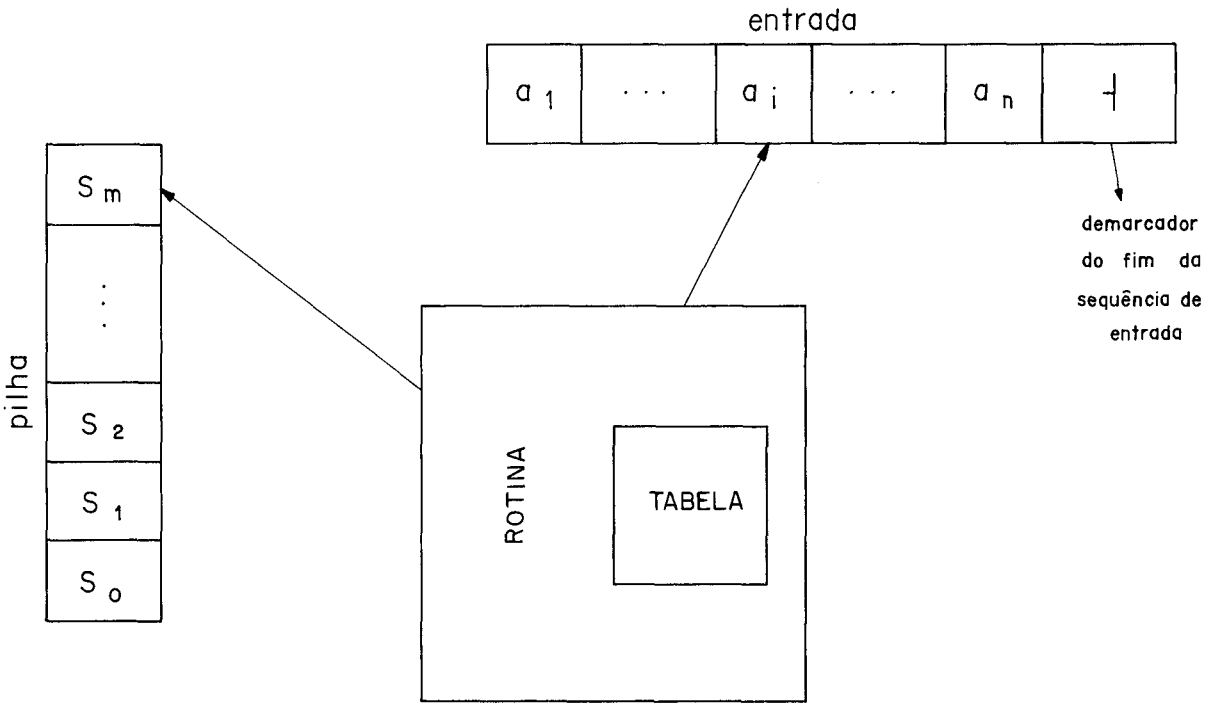
(2) o método de análise sintática é uma particularização do método AVANÇA/REDUZ, podendo portanto ser implementado com o mesmo grau de eficiência deste;

(3) detectam erros sintáticos tão prontamente que nem um símbolo é lido desnecessariamente.

Logicamente, um analisador sintático LR consiste de uma rotina que dirige a análise, baseada em uma tabela. Há diferentes técnicas de geração da tabela de análise LR, entre as quais citaremos duas: SIMPLE LR (SLR(1)), a mais fácil de implementar, e LR(1), a mais poderosa e abrangente apesar de ter im

plementação custosa.

Um analisador LR (SLR(1) ou LR(1), no caso) utiliza uma pilha, a entrada e a tabela para processar a análise:



A entrada é lida, palavra a palavra. A sequência da pilha é formada de S_i, estado do analisador (S_m está no topo), que resume a informação contida abaixo dele na pilha e é usado para guiar as decisões AVANÇA/REDUZ durante a análise, juntamente com o primeiro símbolo da entrada.

Podemos representar o desempenho de um analisador sintático como um par cujo primeiro componente é o conteúdo da pilha e cujo segundo componente é a parte da entrada que ainda não foi lida:

$$(S_0S_1S_2 \ldots S_m,\ a_i\ a_{i+1} \ldots a_n \dashv).$$

O algoritmo de análise sintática é muito simples.

Inicialmente o analisador tem a configuração $(S_0, a_1 a_2 \ldots a_n \dashv)$, onde S_0 é o estado inicial e $a_1 a_2 \ldots a_n$ é a sequência que deve ser analisada. A tabela que comanda o analisador consiste de duas partes: uma função AÇÃO e uma função PROXIMO. A cada processamento do analisador, a rotina que o dirige se comporta da seguinte maneira:

(1) Determina S_m, o estado corrente no topo da pilha;

(2) Determina a_i, a palavra de entrada corrente;

(3) Consulta a tabela AÇÃO, com os dados de (1) e (2), ou seja, AÇÃO(S_m, a_i). Esta entrada da tabela pode ter os seguintes valores:

   (a) AVANÇA S ;

   (b) REDUZ $A \rightarrow \beta$;

   (c) ACEITA;

   (d) ERRO.

As configurações resultantes após cada um dos 4 tipos de processamento são as seguintes:

(1) Se AÇÃO(S_m, a_i) = AVANÇA S , o analisador executa um empilhamento:

   $(S_0 S_1 S_2 \ldots S_m \; S, \; a_{i+1} \ldots a_n \dashv)$;

(2) Se AÇÃO(S_m, a_i) = REDUZ $A \rightarrow \beta$, então o analisador faz uma redução equivalente à seguinte configuração resultante:

   $(S_0 S_1 S_2 \ldots S_{m-r} S, \; a_i a_{i+1} \ldots a_n \dashv)$,

   onde S = PROXIMO(S_{m-r}, A) e r é o comprimento de β. Mais explicitamente, o analisador tira r símbolos da pilha (r estados), deixando no topo da pilha o estado S_{m-r} ;

(3) Se AÇÃO(S_m,a_i) = ACEITA, a análise foi completada com sucesso;

(4) Se AÇÃO(S_m,a_i) = ERRO, o analisador descobriu um erro e chama a rotina de recuperação de erros.

O analisador repete este procedimento até encontrar uma situação de aceitação da sequência ou um erro.

2.3.1. CONSTRUÇÃO DA TABELA SINTÁTICA SLR(1)

Esta seção mostra como construir um analisador sintático do tipo SLR(1) para uma gramática. A idéia central é construir um AFD a partir da gramática e depois transformá-lo em uma tabela de análise sintática SLR(1).

Estabeleceremos inicialmente alguns conceitos necessários à compreensão do método de construção da tabela.

Considere uma gramática BNF livre de contexto G = (N,Σ,P,S)..

DEFINIÇÃO. Suponha que $S \underset{rm}{\overset{*}{\Longrightarrow}} \alpha Aw \underset{rm}{\Longrightarrow} \alpha\beta w$ é uma derivação direita na gramática G . Dizemos que uma sequência γ é um prefixo viável de G se γ é um prefixo de $\alpha\beta$. Isto é, γ é uma sequência que é um prefixo de alguma forma sentencial direita mas que não ultrapassa o fim do trecho reduzido nesta.

DEFINIÇÃO. Um item LR(Ø) - ITEM - de G é uma regra de G com um ponto (.) em alguma posição do lado direito desta.

Exemplo. Seja a regra $A \rightarrow \alpha\beta$. A partir desta regra são gerados os três ítens abaixo:

$A \rightarrow .\alpha\beta$

$A \rightarrow \alpha.\beta$

$A \rightarrow \alpha\beta.$

DEFINIÇÃO. Um estado do analisador é um conjunto de itens, obtido conforme o algoritmo a seguir.

DEFINIÇÃO. Uma coleção canônica LR(Ø) é um conjunto de estados, obtido pelo algoritmo a seguir.

Intuitivamente, um item indica até que ponto de uma regra o analisador já processou, durante a leitura da entrada. Por exemplo, o 2º item, especificado no exemplo acima, indica que já foi detectada na entrada uma sequência derivada de α e que é esperada, a seguir, uma sequência derivável de β .

A coleção canônica LR(Ø) é a base para a construção da tabela de análise sintática SLR(1). Para construir esta coleção, definimos uma gramática BNF livre de contexto aumentada, uma função PROXIMO e a função FECHAMENTO.

DEFINIÇÃO. Se $G = (N,\Sigma,P,S)$ é uma gramática BNF livre de contexto, então G' , a gramática aumentada para G , é definida da seguinte forma: $G' = (N \cup S',\Sigma,P \cup \{S' \rightarrow S\},S')$.

A função FECHAMENTO(I) , sendo I um estado e $G = (N,\Sigma,P,S)$ uma gramática livre de contexto BNF, é calculada da seguinte forma:

```
PROCEDURE FECHAMENTO (I);
BEGIN
  FOR ∀(A → α.Bβ) ∈ I  e ∀ (B → .∂) ∈ P
  DO I:= I ∪ {B → .∂}
END
```

A função PROXIMO(I,X) , onde I é um estado e X é um símbolo da gramática ($X \in N \cup \Sigma$), é definida como sendo o estado correspondente ao fechamento de todos os itens $A \rightarrow \alpha X.\beta$, tal que $A \rightarrow \alpha.X\beta$ pertence a I .

EXEMPLO. Considere a seguinte gramática BNF livre de contexto aumentada:

E' → E

E → E + T

E → T

T → T * F

T → F

F → (E)

F → id

Se I é o estado que contém apenas o item {E'→.E}, então FECHAMENTO(I) contém os seguintes itens:

E' → .E

E → .E + T

E → .T

T → .T * F

T → .F

F → .(E)

F → .id

Considerando I' = {E' → E., E → E. + T}, PROXIMO (I',+) consistirá de:

E → E + . T

T → . T * F

T → . F

F → . (E)

F → . id

Intuitivamente, T → . T * F em I indica que, em algum ponto do processo de análise sintática, espera-se encontrar uma sequência derivável de T na entrada. Já que T → F é uma regra da gramática, espera-se também uma sequência derivável de F neste ponto. Por este motivo, T → . F também é incluído em FECHAMENTO(I).

A construção de C , a coleção canônica de estados LR(Ø), para uma gramática BNF livre de contexto aumentada G', é feita pelo algoritmo a seguir:

ALGORITMO DE CONSTRUÇÃO DA COLEÇÃO CANÔNICA LR(Ø)

Seja G' uma gramática BNF livre de contexto aumentada, e C uma coleção canônica de estados LR(Ø).

```
ALGORITMO COLEÇÃO CANÔNICA LR (G');
BEGIN
  C:={FECHAMENTO({S' → .S})};
  FOR ∀(I ∈ C) e ∀ X ∈ (N ∪ Σ)|PROXIMO(I,X) ≠ Φ
  DO  C:={PROXIMO(I,X)}∪ C
END;
```

EXEMPLO. Considere a gramática do exemplo anterior. A coleção de conjuntos de itens LR(Ø) para esta gramática é:

I_0 : E' → . E
E → . E + T
E → . T
T → . T * F
T → . F
F → . (E)
F → . id

I_1 : E' → E .
E → E . + T

I_2 : E → T .
T → T . * F

I_3 : T → F .

I_4 : F → (.E)
E → . E + T
E → . T
T → . T * F
T → . F
F → . (E)
F → . id

I_5 : F → id .

I_6 : E → E + . T
T → . T * F
T → . F
F → . (E)
F → . id

I_7 : T → T * . F

F → . (E)

F → . id

I_8 : F → (E.)

E → E . + T

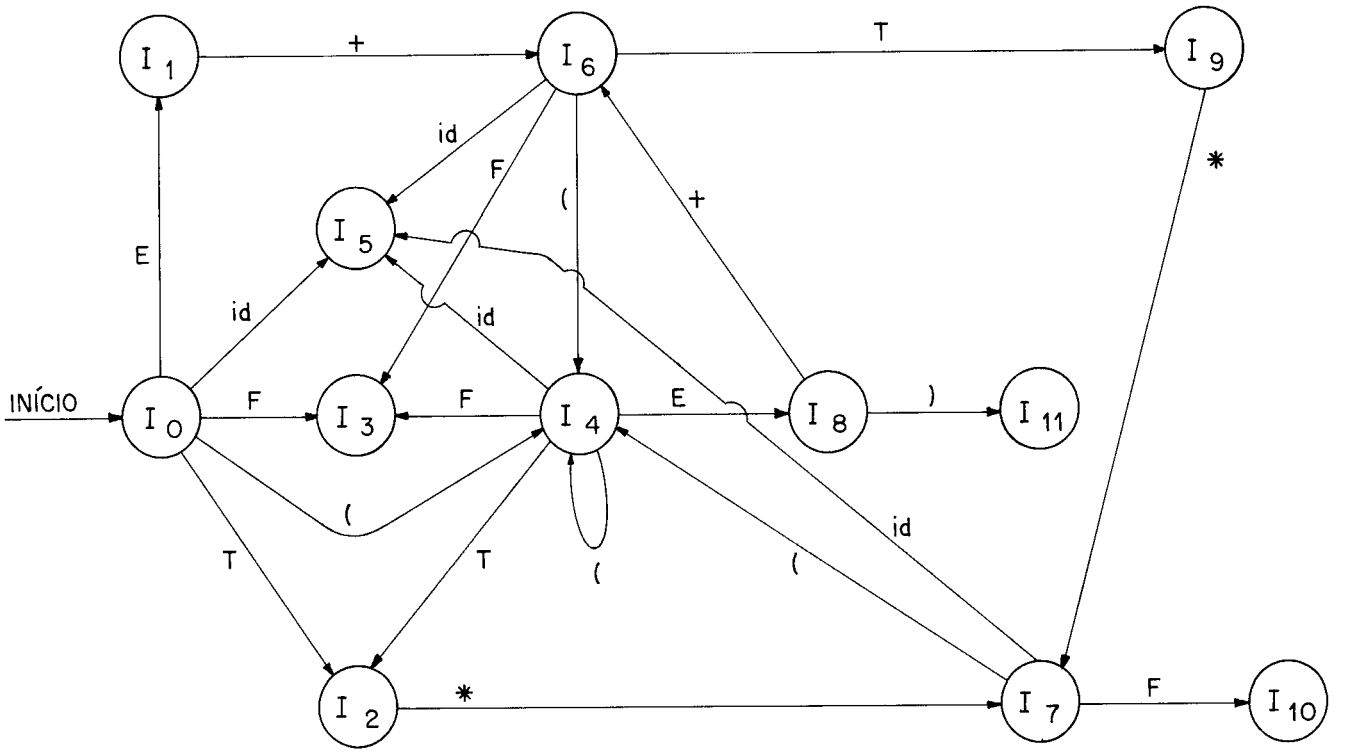
I_9 : E → E + T .

T → T . * F

I_{10}: T → T * F .

I_{11}: F → (E).

O diagrama de transição de um AFD que representa a função PROXIMO para esse conjunto de estados é:



Para toda gramática G , a função PROXIMO da coleção canônica de estados LR(Ø) define um AFD que reconhece os prefixos viáveis de G .

Antes de especificarmos como construir a tabela a partir deste AFD, vamos verificar o significado de cada item e definir as funções SEGUIDOR(A) e COMEÇO(α).

O fato de $A \rightarrow \beta_1.\beta_2$ pertencer a um estado implica na existência da derivação $S' \overset{*}{\Rightarrow} \alpha Aw \Rightarrow \alpha\beta_1\beta_2 w$ e permite indicar se o processamento do analisador deve ser de AVANÇA ou REDUZ quando $\alpha\beta_1$ é representado na pilha do analisador sintático. Em particular, se $\beta_2 \neq \varepsilon$, isto implica que ainda não foi colocado na pilha todo o trecho reduzido; neste caso, o processamento do analisador será AVANÇA. Se $\beta_2 = \varepsilon$, isto significa que o trecho reduzido é $A \rightarrow \beta_1$ e, neste caso, o analisador deve fazer uma REDUÇÃO por esta regra.

DEFINIÇÃO. Seja α uma sequência de símbolos de uma gramática BNF livre de contexto aumentada. COMEÇO(α) = $\{\forall a \in \Sigma \mid \alpha \overset{*}{\Rightarrow} a\beta\}$. Se $\alpha \overset{*}{\Rightarrow} \varepsilon$, então $\varepsilon \in$ COMEÇO(α).

Para calcular COMEÇO(X) para todos os símbolos X da gramática, aplique as seguintes regras até que nenhuma palavra (nem ε) possa ser incluída em qualquer conjunto COMEÇO:

(1) Se X = a então COMEÇO(X) = {a} ;

(2) Se X = A então:

(2.1) Se $(A \rightarrow a\alpha) \in P$ então

COMEÇO(X) = COMEÇO(X) $\cup$ {a} ;

(2.2) Se $(A \rightarrow \varepsilon) \in P$ então

COMEÇO(X) = COMEÇO(X) $\cup$ $\{\varepsilon\}$;

(2.3) Se $(A \rightarrow Y_1\ Y_2 \ldots Y_k) \in P$ então

(2.3.1) Se $Y_1, Y_2, \ldots, Y_{i-1} \in N$ e

$\varepsilon \in$ COMEÇO(Y_J) , $J < i$,

então

COMEÇO(X) = COMEÇO(X) $\cup$ COMEÇO(Y_i).

DEFINIÇÃO. SEGUIDOR(A) = $\{\forall a \in \Sigma \mid S \Rightarrow \alpha A a \beta\}$.

Se A for o símbolo mais à direita em alguma forma sentencial, então -| (o demarcador do fim da sequência de entrada) também deve ser incluído em SEGUIDOR(A).

Para calcular SEGUIDOR(A) para todas as categorias A, aplique as seguintes regras até que não existam mais palavras que possam ser incluídas em qualquer conjunto SEGUIDOR:

(1) -| $\in$ SEGUIDOR(S') ;

(2) Se $(A \rightarrow \alpha B \beta) \in P$ e $\beta \neq \varepsilon$

então SEGUIDOR(B) = SEGUIDOR(B) $\cup$ (COMEÇO(β)-$\{\varepsilon\}$);

(3) Se $(A \rightarrow \alpha B) \in P$ ou

$((A \rightarrow \alpha B \beta) \in P$ e $\varepsilon \in$ COMEÇO(β))

então SEGUIDOR(B) = SEGUIDOR(B) $\cup$ SEGUIDOR(A).

EXEMPLO. Considere a seguinte gramática BNF livre de contexto:

E $\rightarrow$ T E'

E' $\rightarrow$ + T E'

E' $\rightarrow$ ε

T $\rightarrow$ F T'

$T' \rightarrow * \; F \; T'$

$T' \rightarrow \varepsilon$

$F \rightarrow (E)$

$F \rightarrow id$

Então: SEGUIDOR(E) = SEGUIDOR(E') = {),-| }

SEGUIDOR(T) = SEGUIDOR(T') = {+,),-| }

SEGUIDOR(F) = {+,*,),-| }

A forma de construção de uma tabela sintática SLR(1), considerando como entrada a coleção canônica de conjuntos de itens LR(Ø) - C - para uma gramática BNF livre de contexto aumentada G' , é a seguinte:

Seja $C = \{I_0, I_1, \ldots, I_n\}$. Os estados do analisador sintático são $Ø, 1, \ldots, n$ sendo que o estado i é construído a partir de I_i . As ações da análise sintática para o estado i são determinadas da seguinte maneira:

(1) Se $A \rightarrow \alpha.a\beta \in I_i$ e PROXIMO(I,a) = I_j , sendo a uma palavra, então AÇÃO(i,a) = AVANÇA j ;

(2) Se $A \rightarrow \alpha. \in I_i$, então AÇÃO(i,a) = REDUZ $A \rightarrow \alpha$ para todo a pertencente a SEGUIDOR(A);

(3) Se $S' \rightarrow S. \in I_i$, então AÇÃO(i,-|) = ACEITA.

As transições PROX para o estado i são construídas usando a seguinte regra:

(4) Se PROXIMO(I_i,A) = I_j , então PROX(i,A) = j .

As entradas que não forem definidas através da aplicação das regras acima são consideradas condições de ERRO. O estado inicial do analisador é o construído a partir do conjunto de

ítens que contêm o ítem S' → .S .

EXEMPLO . Considerando a gramática

0 : E' → E

1 : E → E + T

2 : E → T

3 : T → T * F

4 : T → F

5 : F → (E)

6 : F → id

e sua coleção canônica de conjuntos de ítens LR(Ø), já especificada no penúltimo exemplo, obtemos a tabela SLR(1) especificada a seguir. As ações representadas na tabela estão codificadas da seguinte forma:

(1) S_i significa AVANÇA e coloque o estado i na pilha;

(2) r_j significa: "reduza pela produção de nº J";

(3) acc significa ACEITA;

(4) uma entrada em branco significa erro.

ESTADO	AÇÃO						PROX		
	id	+	*	(	)	⊣	E	T	F
0	S_5			S_4			1	2	3
1		S_6				acc			
2		r_2	S_7		r_2	r_2			
3		r_4	r_4		r_4	r_4			
4	S_5			S_4			8	2	3
5		r_6	r_6		r_6	r_6			
6	S_5			S_4				9	3
7	S_5			S_4					10
8		S_6			S_{11}				
9		r_1	S_7		r_1	r_1			
10		r_3	r_3		r_3	r_3			
11		r_5	r_5		r_5	r_5			

Observe os procedimentos do analisador, a cada passo, com a entrada id * id + id, usando esta tabela:

	PILHA	ENTRADA
(1)	Ø	id * id + id -|
(2)	Ø id 5	* id + id -|
(3)	Ø F 3	* id + id -|
(4)	Ø T 2	* id + id -|
(5)	Ø T 2 * 7	id + id -|
(6)	Ø T 2 * 7 id 5	+ id -|
(7)	Ø T 2 * 7 F 10	+ id -|
(8)	Ø T 2	+ id -|
(9)	Ø E 1	+ id -|
(10)	Ø E 1 + 6	id -|
(11)	Ø E 1 + 6 id 5	-|
(12)	Ø E 1 + 6 F 3	-|
(13)	Ø E 1 + 6 T 9	-|
(14)	Ø E 1	-|

Os símbolos foram colocados na pilha apenas para visualização.

A tabela de análise sintática constituída das funções AÇÃO e PROX é chamada de Tabela SLR(1) para G . Um analisador sintático LR que usa a tabela SLR(1) para G é chamado de analisador sintático SLR(1) para G . E uma gramática para a qual pode ser construída uma tabela de análise sintática SLR(1) é dita ser SLR(1).

Nem sempre este método produz tabelas sem conflito na função AÇÃO, pois dois itens podem indicar diferentes tipos de procedimento (AVANÇA/REDUZ e REDUZ por i/j) para o mesmo pre

fixo viável.

EXEMPLO. Considere a seguinte gramática BNF livre de contexto:

S → L = R

S → R

L → * R

L → id

R → L

A coleção canônica de estados LR(Ø) para esta gramática é a seguinte:

I_0 : S' → . S

S → .L = R

S → .R

L → . * R

L → . id

R → .L

I_1 : S' → S.

I_2 : S → L. = R

R → L.

I_3 : S → R.

I_4 : L → * . R

R → .L

L → . * R

L → . id

I_5 : L → id .

I_6 : S → L = . R

R → .L

L → . * R

L → . id

I_7 : L → * R .

I_8 : R → L .

I_9 : S → L = R .

Considere I_2 . O 1º item faz com que AÇÃO(2,=) = AVANÇA 6. Já que SEGUIDOR(R) contém = (considere S => L = R => *R = R), o segundo ítem faz com que AÇÃO(2,=) = REDUZ R → L . Portanto, a entrada AÇÃO(2,=) é duplamente definida e de forma conflitante.

Se ocorrem conflitos na geração da tabela SLR(1) para uma determinada gramática, dizemos que a gramática não é SLR(1); e, portanto, não pode ser produzido um analisador sintático SLR(1) para ela.

Para gramáticas onde isso não for possível, uma possível fonte adicional de informação que o analisador pode usar para tomar suas decisões AVANÇA/REDUZ, ao tempo de construção de sua tabela, seriam os próximos k símbolos de entrada em vez de apenas um. Na prática, porém, utilizam-se analisadores apenas com k = 1 devido ao espaço necessário para armazenamento das tabelas.

Nesse caso, outros métodos de análise da família LR poderão ser utilizados.

2.3.2. CONSTRUÇÃO DE TABELAS CANÔNICAS DE ANÁLISE SINTÁTICA LR(1)

Uma gramática que pode ser analisada por um analisador LR que verifica k símbolos de entrada, a cada processamento, é chamada de gramática LR(k).

Intuitivamente, uma gramática não é LR(1) se o analisador sintático desta, em um determinado momento, conhecendo o conteúdo da pilha e o próximo símbolo de entrada, não pode decidir entre AVANÇA/REDUZ ou não pode decidir qual redução fazer.

No método SLR(1) nós vimos que o estado i pede uma redução por $A \to \alpha$ com a palavra de entrada a, se o estado I_i contém o ítem $A \to \alpha$ e $a \in$ SEGUIDOR(A). Em algumas situações, entretanto, quando o estado i aparece no topo da pilha, um prefixo viável $\beta\alpha$ pode estar na pilha mas βA pode não aceitar ser seguido por um a em uma forma sentencial direita. Neste caso, a redução $A \to \alpha$ seria inválida com o símbolo a na entrada. O exemplo de conflito da seção anterior se aplica a este caso.

Nos estados de um analisador sintático LR(1) é possível armazenar mais informação, o que nos permite desconsiderar algumas destas reduções inválidas; que deixam de provocar conflito se ao invés de uma tabela SLR(1) construirmos uma tabela LR(1) para a gramática.

Dividindo os estados quando necessário, podemos fazer com que cada estado do analisador indique quais as palavras da entrada que podem seguir um trecho reduzido α, para as quais há uma redução para A. Incorporamos esta informação

extra no estado redefinindo os ítens, de forma que incluam uma palavra como segundo componente.

DEFINIÇÃO. Um ítem LR(1) tem a forma $[A \to \alpha.\beta, a]$, onde $A \to \alpha\beta$ é uma regra e a é uma palavra ou o delimitador -| .

O segundo componente do ítem LR(1) é chamado de AVANÇO (LOOKAHEAD). O avanço não tem efeito sobre um ítem da forma $[A \to \alpha.\beta, a]$ em que $\beta \neq \varepsilon$, mas um ítem da forma $[A \to \alpha., a]$ indica uma redução por $A \to \alpha$ caso a próxima palavra de entrada seja a . Portanto, a redução $A \to \alpha$ só será feita para as palavras de entrada a tal que $[A \to \alpha., a]$ é um ítem LR(1) no estado do topo da pilha.

O método para construir a coleção de estados LR(1) para uma gramática BNF livre de contexto aumentada é essencialmente o mesmo definido na seção anterior. Ele é especificado a seguir, juntamente com a forma de calcular a função PROXIMO e a operação FECHAMENTO.

Seja I um estado e $G = (N,\Sigma,P,S)$ uma gramática BNF livre de contexto.

```
PROCEDURE FECHAMENTO (I);
BEGIN
  FOR ∀ (A → α.Bβ, a) ∈ I  e ∀ (B → .∂) ∈ P  e
      ∀ b ∈ COMEÇO(βa)  ,  b ∈ Σ
  DO  I:=I ∪ {B → .∂, b}
END
```

A função PROXIMO(I,X), onde I é um estado e X é um símbolo da gramática ($X \in N \cup \Sigma$), é definida como sendo

o estado correspondente ao fechamento de todos os itens $[A \to \alpha X.\beta, a]$, tal que $[A \to \alpha.X\beta, a]$ pertence a I .

ALGORITMO DE CONSTRUÇÃO DA COLEÇÃO DE ESTADOS LR(1)

Seja G' uma gramática BNF livre de contexto aumentada, e C uma coleção de estados LR(1).

```
ALGORITMO COLEÇÃO LR (G');
BEGIN
  C:={FECHAMENTO({S' → .S, -| })};
  FOR ∀ (I ∈ C)  e ∀ X ∈ (N ∪ Σ)|PROXIMO(I,X) ≠ Φ
  DO   C:=PROXIMO(I,X) ∪ C
END;
```

EXEMPLO. Considere a seguinte gramática:

S' → S

S → CC

C → cC

C → d

A coleção de estados LR(1) para esta gramática é:

I_0 : S' → . S , -|
S → . CC , -|
C → . cC , c/d
C → . d , c/d

I_1 : S' → S . , -|

I_2 : S → C.C , -|
C → .cC , -|
C → . d , -|

I_3 : C → c.C, c/d

C → .cC, c/d

C → .d, c/d

I_4 : C → d., c/d

I_5 : S → CC., -|

I_6 : C → c.C, -|

C → .cC, -|

C → .d, -|

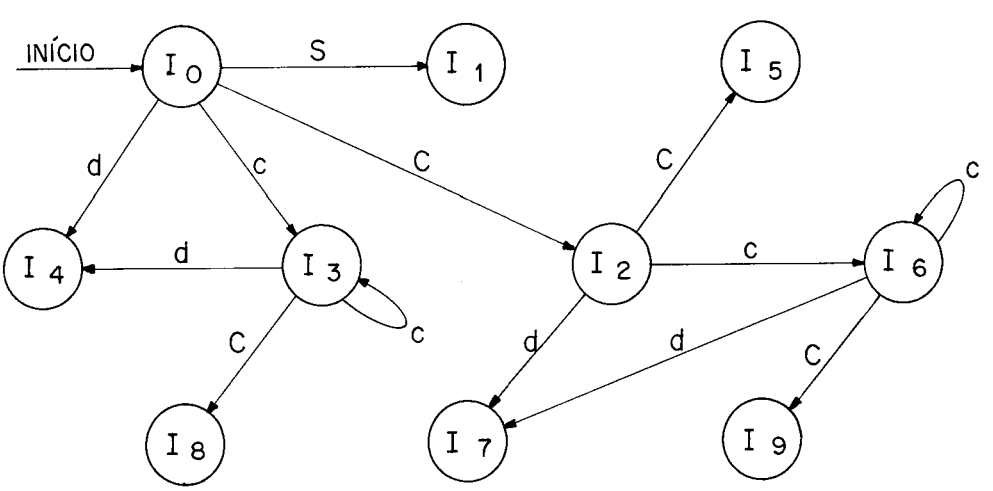
I_7 : C → d., -|

I_8 : C → cC., c/d

I_9 : C → cC., -|

Observe que um ítem do tipo C → cC., c/d é uma forma resumida de representar os ítens C → cC., c e C → cC.,d.

O grafo de transição abaixo mostra os conjuntos de ítens, relacionando-os com os símbolos que geraram sua criação através da função PROXIMO.



A partir dos conjuntos de ítens LR(1) podemos construir a tabela de análise sintática LR(1).

MÉTODO DE CONSTRUÇÃO

(1) Construa $C = \{I_0, I_1, \ldots, I_n\}$, a coleção de conjuntos de ítens LR(1) para G aumentada;

(2) O estado i do analisador sintático é construído a partir de I_i . As ações sintáticas para o estado i são determinadas da seguinte forma:

(a) Se $[A \to \alpha.a\beta, b] \in I_i$ e $PROXIMO(I_i, a) = I_j$, sendo a uma palavra, então

AÇÃO(i,a) = AVANÇA j ;

(b) Se $[A \to \alpha., a] \in I_i$, então AÇÃO(i,a) = REDUZ $A \to \alpha$;

(c) Se $[S' \to S., \dashv] \in I_i$, então

AÇÃO(i, $\dashv$) = ACEITA.

Observação: Se resultar algum conflito da aplicação das regras acima, dizemos que a gramática não é LR(1) e não pode ser criada uma tabela sintática para ela através deste método.

(3) As transições PROX para o estado i são determinadas do seguinte modo:

Se $PROXIMO(I_i, A) = I_j$, então

PROX(i,A) = j ;

(4) As entradas que não foram definidas pela aplicação das regras acima são consideradas condições de ERRO;

(5) O estado inicial do analisador é o construído a partir do conjunto que contém o ítem [S' → . S , -|].

EXEMPLO. Considere a seguinte gramática BNF livre de contexto aumentada:

0 : S' → S

1 : S → CC

2 : C → cC

3 : C → d

Considerando a coleção de estados LR(1) especificada para esta gramática no exemplo anterior, temos a seguinte tabela canônica de análise sintática:

ESTADOS	AÇÃO			PROX	
	c	d	-|	S	C
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4	r3	r3			
5			r1		
6	s6	s7			9
7			r3		
8	r2	r2			
9			r2		

Concluímos assim a apresentação dos métodos de geração de tabelas sintáticas BNF LR(1) e BNF SLR(1). Particularmente importante é o método de geração da tabela para o analisador sintático SLR(1), que terá seus princípios observados no gerador que ora propomos nesta tese, com algumas adaptações necessárias.

3. GRAMÁTICAS LIVRES DE CONTEXTO COM LADO DIREITO REGULAR

3.1. Introdução

Apresentaremos uma nova classe de gramáticas que geram linguagens livres de contexto, denominadas GRAMÁTICAS COM LADO DIREITO REGULAR (RRP) |LaLonde,75|, em que o lado direito das produções é um conjunto regular. E introduziremos uma classe de analisadores sintáticos, denominada RRP SLR(Ø,1), para este tipo de gramática.

3.2. Definição de Gramáticas Livres de Contexto RRP

Esta nova classe de gramáticas possibilita um certo controle na estruturação de analisadores sintáticos para a linguagem, devido à presença de um conjunto regular no lado direito de suas produções.

DEFINIÇÃO. Uma gramática livre de contexto RRP G é uma quádrupla (N,Σ, P,S), onde:

(1) N é um conjunto finito de categorias gramaticais;

(2) Σ é um conjunto finito de palavras, com $N \cap \Sigma = \Phi$;

(3) P é um conjunto finito de regras sintáticas da forma

$$(A_i \rightarrow \pi_i)$$

onde π é um conjunto regular sobre $(N \cup \Sigma)$;

(4) S é o símbolo inicial, $S \in N$.

É conveniente representarmos uma gramática livre de contexto RRP através de uma lista de regras. Neste caso:

(1) N é o conjunto de símbolos presentes no lado esquerdo das regras;

(2) Σ é o conjunto dos demais símbolos;

(3) S é o lado esquerdo da primeira regra da lista;

(4) P é representado pela lista de regras.

Omitiremos, por simplicidade, a menção livre de contexto.

3.3. Formatos de Gramáticas RRP

Qualquer gramática em que o lado direito das regras é uma descrição de um conjunto regular, tais como autômatos finitos, diagramas de transição e expressões regulares, é considerada uma representação de uma gramática RRP.

EXEMPLO. Apresentamos abaixo a mesma gramática RRP descrita em diferentes formatos |LaLonde,75| e |Simone,80|:

(a) Formato Expressão Regular

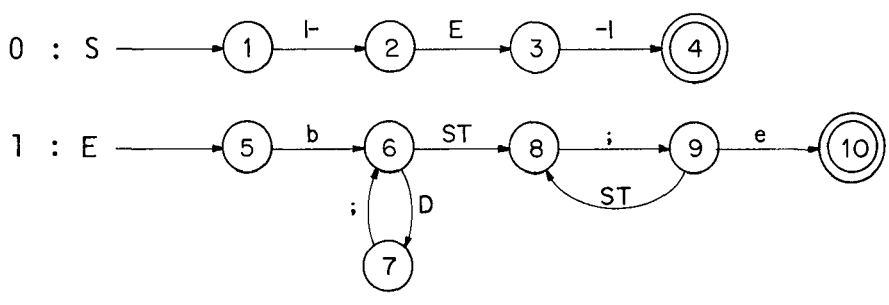
0 : S → |- E -|

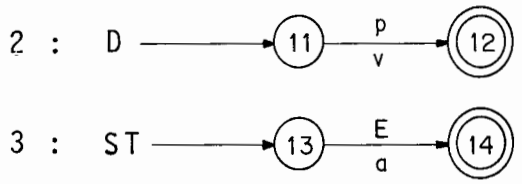
1 : E → b(D;)*(ST;)$^{+}$ e

2 : D → p|v

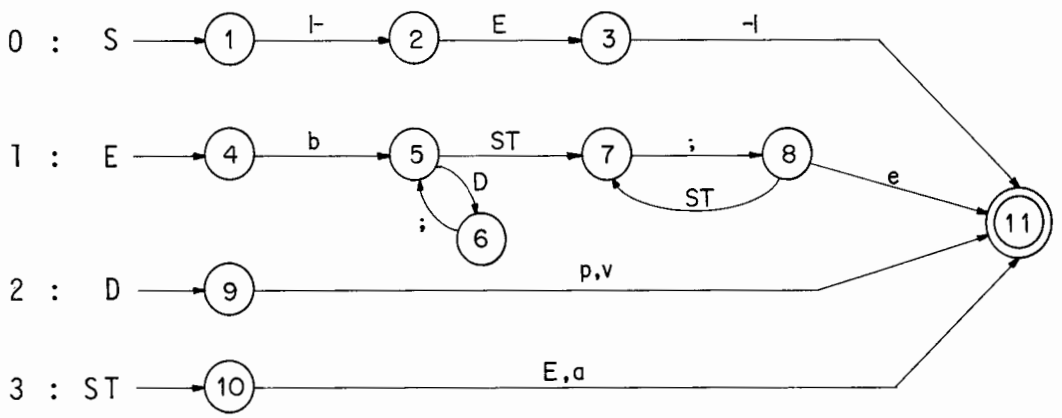
3 : ST→ E|a

(b) Formato Diagrama

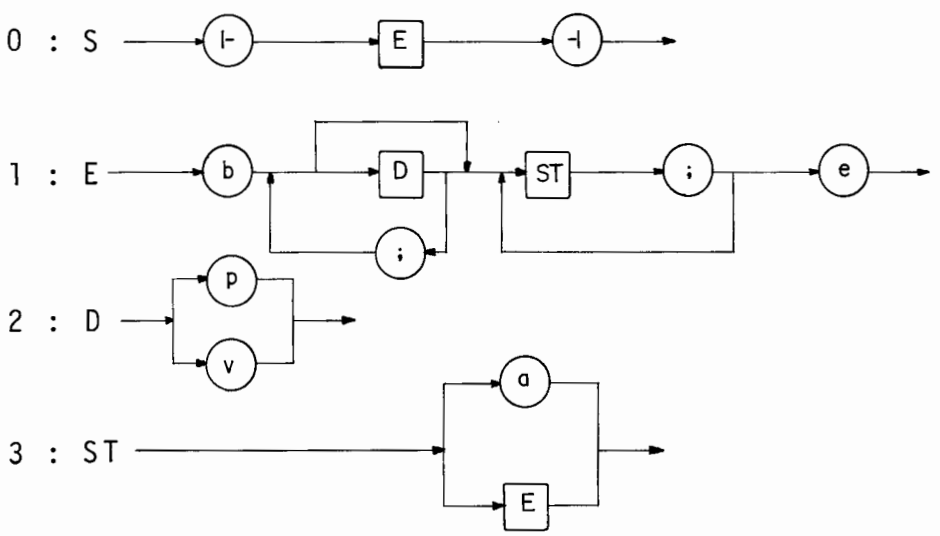




(c) Formato Diagrama Compactado



(d) Formato Diagrama Sintático |WIRTH,75|



(e) Formato Autômato Finito (AF)

0 : S → ({1,2,3,4},V,R_1,{1},{4})

1 : E → ({5,6,7,8,9,10},V,R_2,{5},{10})

2 : D → ({11,12},V,R_3,{11},{12})

3 : ST→ ({13,14},V,R_4,{13},{14})

onde: $V = \{|\text{-}, \text{-}|, b, ;, e, p, v, a, S, E, D, ST\}$

$R_1 \supset \{((1,|\text{-}),\{2\}), ((2,E),\{3\}), ((3,\text{-}|),\{4\})\}$

$R_2 \supset \{((5,b),\{6\}),((6,D),\{7\}), ((6,ST),\{8\}),$
$((7,;),\{6\}),((8,;),\{9\}), ((9,ST),\{8\}),$
$((9,e),\{10\})\}$

$R_3 \supset \{((11,p),\{12\}),((11,v),\{12\})\}$

$R_4 \supset \{((13,E),\{14\}),((13,a),\{14\})\}$

Nesta tese, nossa opção em termos de representação do conjunto de regras sintáticas de uma RRPG abrangerá:

(1) Expressões Regulares (ER) no lado direito das regras, como forma de representação externa da gramática. Esta escolha deveu-se à facilidade de escrever a gramática desta forma e à facilidade de transformá-la em uma forma de representação interna (AF) facilmente tratável pela máquina. Formalmente,

π_i é representado por ER sobre $(N \cup \Sigma)$.

Em |Simone,80| são encontrados algoritmos que transformam os formatos.

(2) Autômatos Finitos Determinísticos representando o lado direito das produções, como forma de representação interna da gramática. Esta escolha revelou-se a mais adequada para os algoritmos de geração das tabelas de análise sintática da gramática. Formalmente,

π_i é representado por $M_i=(K_i,(N \cup \Sigma),\delta_i,Q_i,F_i)$

onde K_i : conjunto finito de estados

δ_i : função de transição

$(K_i \times (N \cup \Sigma)) \rightarrow K_i$

Q_i : estado inicial, $Q_i \in K_i$

F_i : conjunto de estados finais, $F_i \subseteq K_i$.

Assumiremos ainda que:

a) M_i é um AFD mínimo para todo i ;

b) se $i \neq j$ então $K_i \cap K_j = \Phi$, ou seja, não há estados comuns entre os autômatos de cada regra sintática.

3.4. Derivação em gramáticas RRP

O conceito de derivação em gramáticas RRP é uma extensão do conceito de derivação em gramáticas BNF livres de contexto.

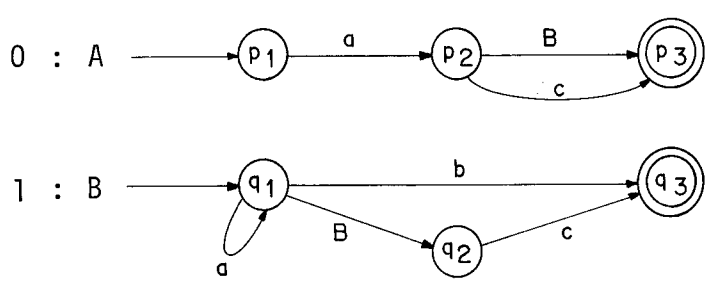
Considere uma gramática RRP $G = (N,\Sigma,P,S)$, onde o conjunto de regras é representado por autômatos finitos determinísticos.

Uma derivação (=>) em G é definida por:

$\alpha A \beta \Rightarrow \alpha \partial \beta \Leftrightarrow (A \rightarrow \pi) \in P$ e $\partial \in \pi$.

Se $(A \rightarrow \pi) \in P$ e $\partial \in \pi$ diremos que $A \rightarrow \partial$ é uma regra derivada de G .

EXEMPLO. Considere a seguinte gramática RRP G com os AFDs em diagrama:



Já que $p_1 \xrightarrow{aB} p_3$ e $p_1 \xrightarrow{ac} p_3$, podemos dizer que $A \rightarrow aB$ e $A \rightarrow ac$ são regras derivadas de G ; similarmente, podemos escrever que $A \Rightarrow aB$ e $A \Rightarrow ac$. Da mesma forma, podemos escrever que $B \Rightarrow b$. Já que $q_1 \xrightarrow{Bc} q_3$, $q_1 \xrightarrow{aBc} q_3$, $q_1 \xrightarrow{aaBc} q_3$,..., podemos também escrever que $B \Rightarrow Bc$, $B \Rightarrow aBc$, $B \Rightarrow aaBc$, ...

Uma observação importante a fazer, aproveitando o exemplo acima, é que o lado direito de uma regra derivada de uma gramática RRP é de tamanho variável . Consequentemente, pode haver um número ilimitado de diferentes regras derivadas para uma única regra da gramática.

Esta alteração no conceito de derivação é o problema fundamental da geração de um analisador sintático ascendente para este tipo de gramática; já que na derivação de uma gramática RRP, uma categoria é substituída por uma sequência de palavras determinada pelo conjunto regular que representa o lado direito da regra sintática.

A principal vantagem das RRPGs é a eliminação de categorias, transformando recursões, na notação BNF, em iterações no interior dos autômatos. Em teoria, apenas as categorias "self-embedded" - ou seja, $A \overset{*}{\Rightarrow} \alpha A \beta$ com α e β não nulos - não podem ser eliminadas. As RRPGs são consideradas como uma descrição mais clara, concisa e natural da sintaxe das linguagens de programação do que a notação BNF. Na prática, o número de regras sintáticas não é sempre mínimo, devido à necessidade de preservar precedências ou diminuir o tamanho do analisador.

Sua maior clareza decorre da eliminação de categorias que não pertencem à linguagem em si, mas apenas representam relações entre suas palavras.

3.5. Definições básicas

(1) Uma RRPG é reduzida se

a) é fértil: $\forall A_i \ , \ A_i \overset{+}{\Rightarrow} w \ , \ w \in \Sigma^*$

b) é acessível: $\forall A_i \ , \ S \overset{*}{\Rightarrow} \alpha A_i \beta$

c) é acíclica: $\forall A_i \ , \ A_i \overset{+}{\neq\Rightarrow} A_i$

(2) Uma RRPG é unívoca se tem uma só regra sintática por categoria, isto é:

$\forall i, j \ , \ A_i \neq A_j$

(3) RRPG transformada:

$$G = (N,\Sigma,P,S) \Rightarrow G_{transf} = (N \cup \{S'\} \ , \ \Sigma \cup \{\dashv\} \ , \ P \cup \{S' \rightarrow \{S \dashv\}\} \ , \ S' \ )$$

onde $S' \notin N$ e $\dashv \notin \Sigma$

(4) Dada a RRPG G , G' será sua normalizada se G' é, simultaneamente, reduzida, transformada e unívoca, e

$w \in L(G) \Leftrightarrow w \dashv \in L(G')$.

3.6. Análise Sintática de Gramáticas RRP

Para fazer a análise sintática das gramáticas RRP, desenvolvemos um analisador ascendente, denominado RRP SLR(Ø,1), que pode ser considerado uma generalização do analisador BNF SLR(1); inclusive com o correspondente grau de eficiência e simplicidade.

A análise sintática se processa observando o mesmo modelo básico: uma sequência de operações de avanço e redução. Vamos considerar, inicialmente, que estes movimentos são efetuados por duas máquinas acopladas: uma máquina de LERPARAFRENTE que lê e empilha a sentença até chegar ao final direito do trecho reduzido (handle); e uma máquina de LERPARATRAS que desempilha até detectar seu início (final esquerdo) e retorna a categoria correspondente. O controle do analisador oscila entre estas máquinas até que a LERPARAFRENTE aceita a sentença ou acusa um erro.

3.6.1. Determinação do final direito do trecho reduzido

Para gramáticas RRP a máquina LERPARAFRENTE pode ser construída de forma similar a gramáticas BNF. Os trabalhos de |LaLonde,75| e |LaLonde,79| desenvolvem a teoria necessária para tanto.

3.6.1.1. Máquina característica RRP LR(Ø,Ø)

Um ítem LR(Ø) é uma regra com uma marca do lado direito, indicando o prefixo já empilhado. Como assumimos que não há estados comuns entre autômatos de regras distintas, um estado será uma representação concisa de um ítem LR(Ø).

Dada G, RRPG normalizada, será sempre possível construir sua máquina característica pelo algoritmo a seguir.

Formalmente, dada $G = (N,\Sigma,P,S)$,

$M\emptyset(G) = (K, (N \cup \Sigma), \Delta, q_0, F)$

onde K : conjunto de tabelas LR(Ø,Ø)

$N \cup \Sigma$: alfabeto

Δ : função de transição

$(K \times (N \cup \Sigma)) \rightarrow K$

$q_0 \in K$

$F \subseteq K$

ALGORITMO CONSTRUTOR MØ(G)

ENTRADA : G, RRPG normalizada, formato AF

SAÍDA : MØ e os conjuntos

$NUCLEO_i$: conjunto de estados dos autômatos de G

$FECHAMENTO_i$: conjunto de estados dos autômatos de G

```
|1| (*criação do estado inicial de MØ*)
    Faça I = 1   (*em análise*)
       e J = 1   (*ponteiro para novos*)
    Faça NUCLEO[I]= {Q0}  (*estado inicial da*)
       e J = J+1 ;        (*regra de   S' *)
```

|2| Enquanto $I \leq J$ faça:

|2.1| (* fechamento *)

Faça FECHAMENTO[I] =

$\{Q_k \mid q \in (NUCLEO[I] \cup FECHAMENTO[I])$

$e \ \exists \ \delta(q,A_k)\}$

(* note que $A \in N$ *)

|2.2| (* AVANÇO *)

Para cada $X \in (N \cup \Sigma)$ faça:

|2.2.1| NUCLEO[J] =

$\{s \mid q \in (NUCLEO[I] \cup FECHAMENTO[I])$ e

$\delta(q,X) = s\}$

|2.2.2| Se $\exists \ K < J \mid NUCLEO[K] = NUCLEO[J]$

então faça $P = K$

senão faça $P = J$ e $J = J + 1$;

|2.2.3| Faça $\Delta(I,X) = P$

|2.3| Faça $I = I + 1$;

EXEMPLO.

G = (N,Σ,P,S') normalizada

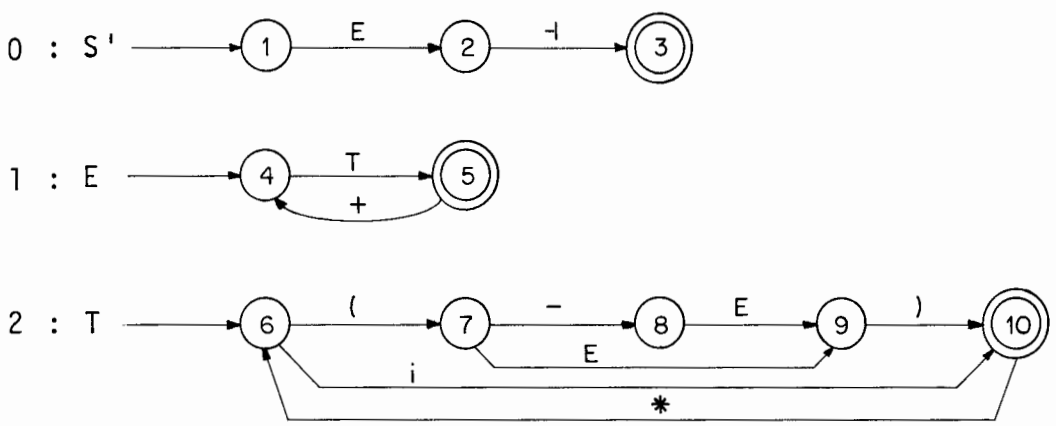


Fig.1. G em formato autômato finito determinístico.

MÁQUINA CARACTERÍSTICA RRP LR(Ø,Ø)

K	δ									NUCLEO	FECHAMENTO
	⊣	(	)	+	-	i	*	E	T		
q_0=1		4				5		2	3	1	4,6
2	6									2	
3				7						5	
4		4			8	5		9	3	7	4,6
5							10			10	
6										3	
7		4				5			3	4	6
8		4				5		9	3	8	4,6
9			5							9	
10		4				5				6	

3.6.1.2. Máquina LERPARAFRENTE RRP SRL(Ø,1)

Inicialmente, definimos o conjunto SEGUIDOR como:

$$SEGUIDOR(A) = \{a \mid S' \overset{*}{\Rightarrow} \alpha A a \beta\}$$

Para que G seja RRP SLR(Ø,1) é necessário que obedeça, primeiramente, às condições indicadas abaixo. Note que são condições parciais , pois estamos considerando apenas a construção da máquina LERPARAFRENTE. Considerando que:

$\forall q \in K$ em MØ e

$\forall p,r \in (NUCLEO[q] \cup FECHAMENTO[q])$

$p \in F_i$ e $r \in F_j$,

$p \neq r$, $i \neq j$.

Sendo A_i e A_j dois itens completos de uma coleção de itens, as condições parciais para que G seja RRP SLR(Ø,1) são:

1 - SEGUIDOR[A_i] ∩ SEGUIDOR[A_j] = Φ

2 - a ∈ (SEGUIDOR[A_i] => ∄ Δ(q,a)

O caso 1 é o conflito "redução-redução" e o caso 2 é o conflito "avanço-redução".

Construa a máquina LERPARAFRENTE SLR(Ø,1), supondo satisfeitas as condições acima, da seguinte forma:

a) AÇÃO : (K x Σ) → {AVANÇA,REDUZ,ACEITA,ERRO}

i. Δ(q,a) = q_1 => AÇÃO[q,a] = AVANÇA

ii. p ∈ (NUCLEO[q] ∪ FECHAMENTO[q])
p ∈ F_i
a ∈ SEGUIDOR[A_i]
=> AÇÃO[q,a] = REDUZ

iii. p ∈ (NUCLEO[q] ∪ FECHAMENTO[q])
p ∈ F_0 , regra inicial
=> AÇÃO[q,-|] = ACEITA

iv. AÇÃO[q,a] = ERRO, nos demais casos.

b) PROX: (KxN) → K

i. Δ(q,A) = q_1 => PROX[q,A] = q_1

c) AVANÇAREDUZ : (K x Σ) → {1,2,...,n}

i. AÇÃO[q,a] = AVANÇA
Δ(q,a) = q_1 => AVANÇAREDUZ[q,a] = q_1

ii. AÇÃO[q,a] = REDUZ
p ∈ (NUCLEO[q] ∪ FECHAMENTO[q])
p ∈ F_i
a ∈ SEGUIDOR[A_i]
=> AVANÇAREDUZ[q,a] = i

EXEMPLO. Utilizando a gramática G do exemplo anterior.

MÁQUINA LERPARAFRENTE RRP SLR(Ø,1)

K	AÇÃO/AVANÇAREDUZ							PROX		NUCLEO	FECHAMENTO
	-|	(	)	+	-	i	*	E	T		
1		4				5		2	3	1	4,6
2	6									2	
3	R1		R1	7						5	
4		4			8	5		9	3	7	4,6
5	R2		R2	R2			10			10	
6	AC									3	
7		4				5			3	4	6
8		4				5		9	3	8	4,6
9			5							9	
10		4				5				6	

SEGUIDOR: S' : Φ

E : {-| ,)}

T : {-| ,) , +}

Foram utilizadas as seguintes abreviaturas:

REDUZ : R

AVANÇA: omitido

ACEITA: AC

ERRO : em branco.

O conjunto de ítens associado a cada tabela SLR(Ø,1) é representado por:

$$\text{NUCLEO} \in \mathcal{P}\left(\bigcup_j K_j\right) \quad \text{e}$$

$$\text{FECHAMENTO} \in \mathcal{P}\left(\{Q_1, Q_2, \ldots, Q_p\}\right).$$

Note que apenas estados iniciais pertencem a FECHAMENTO. As ações de redução são colocadas conforme o método SLR(1). O marcador de fim de sentença "-| " é tratado da maneira usual. Após contruída a máquina LERPARAFRENTE, é indistinguível se a gramática original é RRP ou BNF. Outros tipos de máquina LERPARAFRENTE da família LR (LR(K), LR(1), LR(Ø),LALR(1)) podem ser construídas de maneira análoga |Simone,82|. Apenas o número de estados será distinto em cada caso.

TEOREMA.

A máquina LERPARAFRENTE RRP SLR(Ø,1) permite determinar o final direito do trecho reduzido.

DEM. Extensão trivial do método BNF SLR(1), com demonstração em |Aho & Ullman,72|.

3.6.1.3. Funcionamento da máquina LERPARAFRENTE RRP SLR(Ø,1)

O funcionamento desta máquina é ligeiramente diferente do funcionamento usual de máquinas LERPARAFRENTE para gramáticas BNF da família LR. Como poderá ser visto no algoritmo a seguir, a tabela corrente da análise sintática não está no topo da pilha. Ela é denominada ESTADOATUAL e só é empilhada quando a ação ESTADOATUAL X ENTRADA determinar um empilhamento; neste ponto, a tabela determinada por esta entrada da função AÇÃO é colocada em ESTADOATUAL. O motivo desta alteração será

justificado mais adiante.

ALGORITMO

Utiliza uma pilha onde são armazenados os números das tabelas utilizadas. A tabela corrente é denominada ESTADOATUAL.

w é a sentença a ser analisada.

A operação LEIA coloca a primeira palavra de w ainda não lida em ENTRADA.

```
|1| Faça ESTADOATUAL = 1
       e LEIA;
|2| Caso AÇÃO[ESTADOATUAL, ENTRADA] seja:
    "AVANÇA" : push (ESTADOATUAL);
               Faça ESTADOATUAL = AVANÇAREDUZ[ESTADOATUAL,ENTRADA];
               LEIA;
    "REDUZ"  : Faça I = AVANÇAREDUZ[ESTADOATUAL,ENTRADA];
               Emita I ; (* nº da regra usada *)
               Chame a máquina LERPARATRAS;
                 (* esta máquina alterará a pilha e o ESTADOATUAL *)
               push(ESTADOATUAL); (* se for o caso *)
               Faça ESTADOATUAL = PROXIMO[ESTADOATUAL, A_I];
    "ACEITA" : Emita "ACEITO" e PARE;
    "ERRO"   : Emita "ERRO" e PARE;
    Repita o passo 2.
```

3.6.2. Determinação do final esquerdo do trecho reduzido

A máquina de LERPARATRAS BNF SLR(1) apenas desempilha um determinado número de tabelas - igual ao comprimento fixo da regra - e retorna a categoria reduzida. Mas uma regra RRP corresponde a um número ilimitado de regras derivadas com comprimento variável. Isto significa que o final esquerdo do trecho reduzido pode estar a uma distância arbitrária de seu final direito.

Vemos portanto que, na análise sintática ascendente de gramáticas RRP, a detecção da presença de um trecho reduzido no topo da pilha não oferece problemas; pode ser utilizado qualquer analisador da família LR. A dificuldade que estas gramáticas apresentam é a indeterminação do tamanho destes trechos e, portanto, a determinação de seu início (final esquerdo).

Este problema foi o que dificultou a especificação de um analisador sintático ascendente para as gramáticas RRP e diversas técnicas foram desenvolvidas para solucioná-lo.

|DeRemer,70| sugere o uso de uma máquina de controle para o desempilhamento. Posteriormente, entretanto, optou por transformar cada regra RRP em uma gramática regular adicional, retornando dessa forma à notação BNF (|DeRemer,80|e |DeRemer,81|).

|LaLonde,75| e |LaLonde,79| desenvolve a teoria completa do método LR(m,k), onde k é o comprimento do avanço ("lookahead") sobre a sentença e m é o comprimento do recuo ("lookback") sobre a pilha, além do trecho reduzido. Isto significa a definição de uma classe de gramáticas RRP que possuem

analisador sintático determinístico para valores definidos de k e m . Entretanto, esse analisador é complexo, ocupa muito espaço para as tabelas LERPARAFRENTE e LERPARATRAS, e sua implementação, portanto, não é simples.

|Madsen & Kristensen,76| definem um método análogo para o que denominaram GRAMÁTICAS LIVRES DE CONTEXTO ESTENDIDAS (ECFGs), que admitem expressões regulares como lados direitos das regras. O analisador é construído diretamente a partir das expressões regulares, mas seu método de desempilhamento também necessita de recuo sobre a pilha.

A partir do processo generalizado proposto por |LaLonde,75| e das sugestões de |DeRemer,70|, basicamente, definimos uma classe de gramáticas que denominamos RRP SLR(Ø,1), ou seja, sem recuo, e construímos um analisador sintático adequado a esta classe; satisfazendo as exigências de simplicidade e pequeno tamanho, necessárias para equiparar o método, em eficiência, ao BNF correspondente. As restrições impostas à gramática, para que seja RRP SLR(Ø,1), são suficientemente leves para permitir o tratamento das linguagens de programação.

O método aqui proposto utilizou ao máximo a propriedade da classe de analisadores sintáticos LR, que assegura a presença apenas de trechos reduzidos corretos na pilha; daí advém a simplicidade de nosso método de desempilhamento, que dispensa recuo sobre a pilha além do trecho reduzido.

TEOREMA.

O final esquerdo do trecho reduzido através da regra j será uma tabela I tal que: $Q_j \in$ FECHAMENTO[I]

Dem. Trivial pelo simples acompanhamento do construtor da máquina LERPARAFRENTE. A tabela que marcará o final esquerdo do trecho reduzido deve conter necessariamente o estado inicial da regra, pela qual se faz a redução, em seu FECHAMENTO.

Suponha, inicialmente, que estamos acionando uma máquina LERPARAFRENTE RRP SLR(Ø,1) da maneira usual, isto é, todas as tabelas usadas durante a fase de avanço são empilhadas. Note que qualquer tabela j tal que:

$$\text{FECHAMENTO } j = \Phi \qquad (1)$$

não necessita ser empilhada pois não pode corresponder a um começo de trecho reduzido.

Entretanto, toda tabela deve ser empilhada pois é a informação necessária para se determinar a regra derivada, comandar a tradução ou permitir recuperação de erros. Mas, por enquanto, vamos nos concentrar apenas na determinação do início do trecho reduzido.

O processo que pretendemos criar para determinação do final esquerdo do trecho reduzido faz uso da propriedade descrita no teorema acima e apresenta grande simplicidade:

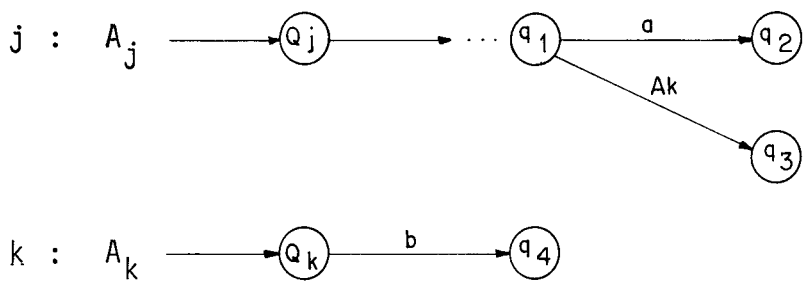
a) ao se determinar uma REDUÇÃO, sabemos a regra usada e bastará descer pela pilha até encontrarmos uma tabela cujo FECHAMENTO contenha o estado inicial da regra;
b) não há necessidade de empilharmos todas as tabelas. Apenas as que tenham FECHAMENTO $\neq \Phi$.

Em particular, não há necessidade de empilharmos

sempre uma tabela que tenha FECHAMENTO $\neq \Phi$, como veremos a seguir.

3.6.2.1. A função EMPILHA

Nosso objetivo é empilhar apenas tabelas que representem começo de trecho reduzido. Como empilhamento é uma propriedade de transições e não de tabelas, a condição FECHAMENTO $\neq \Phi$ é necessária mas não suficiente. Veja-se por exemplo:



Se a tabela atual é i tal que:

$q_1 \in NUCLEO_i$

a transição b implica em empilhar i , enquanto que a transição a implica no oposto.

TABELA	NUCLEO	FECHAMENTO
i	q_1	Q_k
j	q_3	----
k	q_2	----
ℓ	q_4	----

Note que a tabela I dá origem às seguintes ações da máquina LERPARAFRENTE:

AÇÃO[I,a] = AVANÇA j

AÇÃO[I,b] = AVANÇA ℓ

AÇÃO[I,A_k]= PROX[I,A_k]

Quando a transição se dá de q_1 para q_3 (AÇÃO[I,a]= AVANÇA j) a tabela i não deve ser empilhada pois não representa um início de regra. Apenas as transições derivadas de um FECHAMENTO serão empilhadas. Proporemos então uma função:

EMPILHA : (K x (N U Σ)) → {verdadeiro, falso, indeterminado}

onde K é o conjunto de tabelas SLR(Ø,1),

"↓" indicará verdadeiro,

" " indicará falso e "#" indicará indeterminado

cuja definição será dada mais adiante.

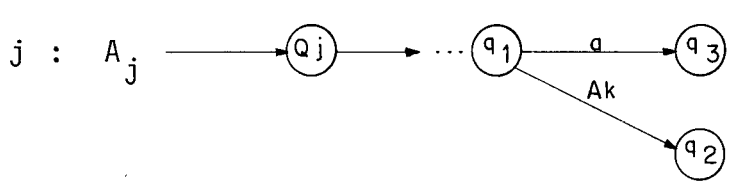
Devido à introdução desta função para comandar o empilhamento no método de análise sintática aqui proposto, é que as funções AÇÃO E PROX foram correspondentemente alteradas de modo que:

AÇÃO[I,a] = AVANÇA j => ESTADOATUAL := j

PROX[I,a] = j => ESTADOATUAL := j

O exemplo anterior não prevê todas as situações que a função EMPILHA tem que tratar. Veremos a seguir outros casos, e como a função EMPILHA se comporta em cada um deles.

Pode ocorrer, por exemplo, o conflito "empilhar ou não empilhar". Isto acontece sempre que atingimos a fronteira entre as gramáticas LL(1) e LR(1), conforme o exemplo abaixo:



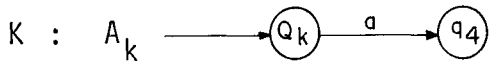


TABELA	NUCLEO	FECHAMENTO
i	q_1	Q_k
j	q_3, q_4	....
k	q_2	....

AÇÃO[i,a] = j

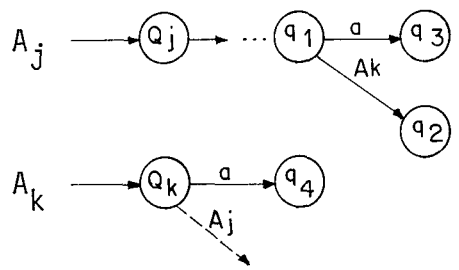
AÇÃO[i,A_k] = k

No caso geral, teremos $A_k \overset{*}{\Rightarrow} a\alpha$ em vez de $A_k \Rightarrow a\alpha$.

Nessas condições nossa decisão será EMPILHAR o ESTADOATUAL, pois poderemos ter uma redução pela regra K . Note que uma gramática RRP onde esse tipo de conflito não acontece será também RRP LL(1).

Esta decisão de empilhamento não trará maiores problemas pois, se for feita a redução pela regra j (significando transição por q_1 e q_3) teremos apenas empilhado desnecessariamente uma tabela a mais.

Há, entretanto, um caso particular que deve ser considerado: quando a tabela empilhada desnecessariamente possui Q_j pertencente ao seu fechamento, ou seja:



e $A_k \overset{*}{\Rightarrow} A_j\alpha$

TABELA	NUCLEO	FECHAMENTO
i	$q_1 \ldots$	Q_k, Q_j ...
j	$q_3, q_4 \ldots$	...
k	$q_2 \ldots$	

No instante de efetuarmos a redução, desempilharíamos até encontrar uma tabela tal que $Q_j \in$ FECHAMENTO. Mas se a transição fosse feita por q_1 e q_3, isso implicaria em erro. Veremos adiante que, se este caso ocorrer, a gramática não é RRP SLR(Ø,1) e, portanto, não poderá ser tratada pelo gerador aqui proposto.

É característico dos analisadores LR possuir um conjunto de pontos no interior da gramática (correspondentes aos estados do NUCLEO) de onde a análise pode prosseguir. Isto significa apenas que, em determinados momentos, não existe informação suficiente para se determinar a localização exata em que o analisador se encontra dentro da gramática. Esta informação só se completará quando a redução for decidida. Para gramáticas não ambíguas uma única regra derivada - e, portanto, uma única trajetória sobre seu autômato - estará determinada.

Definiremos a função EMPILHA da seguinte forma:

Seja uma tabela i com

$NUCLEO_i = \{q_1, q_2, \ldots, q_n\}$ e

$FECHAMENTO_i = \{Q_1, Q_2, \ldots, Q_m\}$

Note que o NUCLEO e o FECHAMENTO são conjuntos distintos.

a) EMPILHA(i,X) = "↓" <=> $Q_k \in FECHAMENTO_i$ (2)

$X \in (N \cup \Sigma)$

$\delta_k(Q_k,X) = r$

$(\nexists q \in NUCLEO_i \mid \delta_j(q,X) = s$

ou

$Q_j \notin FECHAMENTO_i)$

b) EMPILHA(i,X) = "#" <=> $Q_k \in FECHAMENTO_i$ (3)

$X \in (N \cup \Sigma)$

$\delta_k(Q_k,X) = r$

$(\exists q \in NUCLEO_i \mid \delta_j(q,X) = s$

e

$Q_j \in FECHAMENTO_i)$

c) EMPILHA(i,X) = " " , em qualquer outro caso. (4)

O teorema a seguir esclarece precisamente os objetivos desta definição.

TEOREMA.

Toda gramática G , que possue máquina LERPARAFRENTE SLR(1) e para a qual a função EMPILHA não assume o valor "indeterminado" é SLR(Ø,1).

Demonstração.

Utilizaremos a seguinte estratégia de desempilhamento:"se AÇÃO[i,a] = "reduz j" , desempilhe até encontrar t | $Q_j \in FECHAMENTO_t$" . Esta estratégia é claramente SLR(Ø,1), pois não utiliza qualquer tabela à esquerda do trecho reduzido.

A prova compõe-se de duas partes:

i) a tabela $t \mid Q_j \in \text{FECHAMENTO}_t$ está na pilha;

ii) se existirem na pilha $t_m \mid Q_j \in \text{FECHAMENTO}_{t_m}$ e $t_n \mid Q_j \in \text{FECHAMENTO}_{t_n}$, apenas a mais próxima do topo corresponde a uma regra derivada.

Parte (i). Supondo que AÇÃO[i,a] ="reduz j", temos que $q \in F_j$ e $q \in (\text{NUCLEO}_i \cup \text{FECHAMENTO}_i)$. Se $q \in F_j$ então $q \in K_j$, pois $F_j \subseteq K_j$. Logo, será alcançado apenas via estado inicial Q_j, ou seja, $\exists\ t \mid Q_j \in \text{FECHAMENTO}_t$ e $\exists\ X \mid \delta_j(Q_j,X) = s$ e X faz parte da forma sentencial. Como $\text{EMPILHA}(t,X) \neq \text{"\#"}$, estão satisfeitas as condições para $\text{EMPILHA}(t,X) = \text{"}\downarrow\text{"}$.

Parte(ii). Supondo que $\text{AÇÃO}[t_o,a] = \text{"reduz j"}$, que o conteúdo na pilha seja $\ldots t_n t_{n-1} \ldots t_m \ldots t_1 t_o$, que $Q_j \in \text{FECHAMENTO}_{t_n}$ e $Q_j \in \text{FECHAMENTO}_{t_m}$. Se t_n correspondesse ao início de trecho a reduzir pela produção j, então $\exists\ (q_{n-1},\ldots,q_m,\ldots,q_1,q_o \in K_j)$

$$\text{e} \quad q_k \in \text{NÚCLEO}_{t_k}, \quad \emptyset \leq k \leq n-1$$

$$\text{e} \quad q_o \in F_j.$$

Em particular, $q_m \in \text{NUCLEO}_{t_m}$ e $Q_j \in \text{FECHAMENTO}_{t_m}$. Como $\text{EMPILHA}(t_m,X) = \text{"}\downarrow\text{"}$, pois t_m está na pilha, então $\nexists X \mid \delta_j(q_m,X) = s$. Logo, t_n não é início de trecho reduzido, pois não corresponde a uma regra derivada.

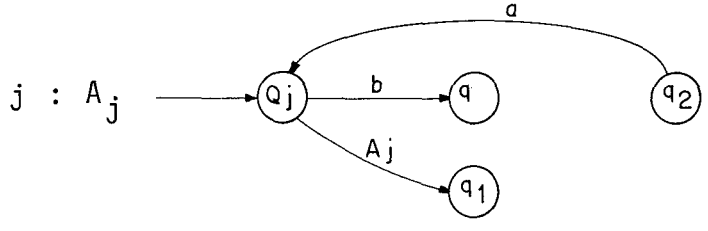
A idéia subjacente à definição da função EMPILHA é que toda tabela correspondente a um trecho reduzido da regra j deve, necessariamente, conter Q_j em seu FECHAMENTO. Preci

samos evitar, entretanto, que qualquer outra tabela com tal condição seja empilhada indevidamente.

Acompanhemos a condição (3) EMPILHA(i,X) = "#" (indeterminado). Temos um estado Q_k em FECHAMENTO$_i$, com $\delta_k(Q_k,X) = r$. Isto deveria significar o empilhamento da tabela i, com o símbolo X, prevendo uma possível redução pela regra k. Entretanto, essa mesma tabela possue em seu NUCLEO um estado q e em seu FECHAMENTO um estado Q_j, ambos pertencentes ao autômato da regra j. E, ainda, q possui transição com X. Ou seja, quando da ocorrência do símbolo X e da tabela i, poderemos estar frente a uma transição no interior, e não no princípio, da regra j. Nesse caso a tabela seria indevidamente empilhada, para o nosso método visado, que pretende considerar como início do trecho reduzido pela regra j a tabela mais próxima do topo que contém Q_j em seu FECHAMENTO. (Note que, possivelmente, $Q_j = Q_k$).

Alguns exemplos esclarecem os casos problemáticos:

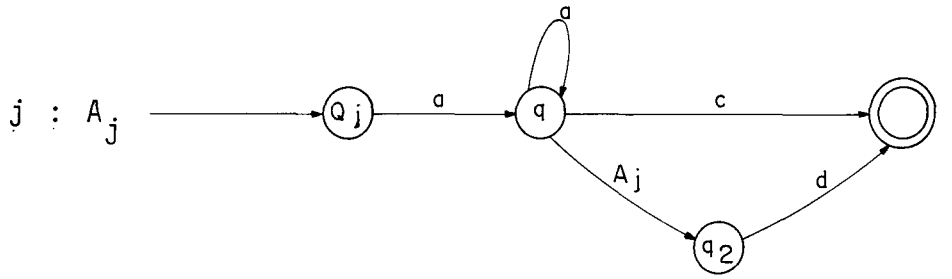
a) recursão à esquerda com ciclo no estado inicial:



Este tipo de regra não pode ser SLR(Ø,1), pois ao desempilharmos uma sequência bbbb... ainda temos que recuar mais um símbolo, devido a uma possível ocorrência de a. A condição (3) será satisfeita pois toda tabela que contém Q_j

em seu NUCLEO (e necessariamente alguma conterá), conterá também Q_j em seu FECHAMENTO;

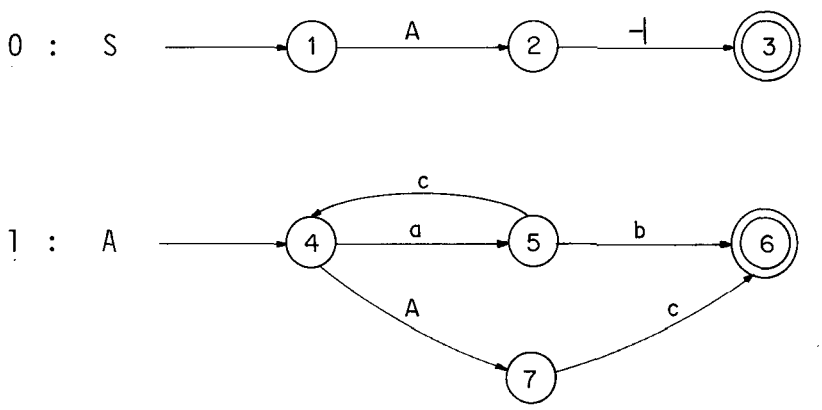
b) recursão "self-embedding" e transição com COMEÇO(A_j):



Este tipo de regra gera a linguagem $a^i aa^j cd^i$. É impossível determinar-se o comprimento da sequência a^j antes de ser conhecido o comprimento da sequência d^i. A condição (3) evita essa classe de regras.

Apresentamos a seguir alguns exemplos destes casos problemáticos.

EXEMPLO 1. Seja a gramática RRP G1:



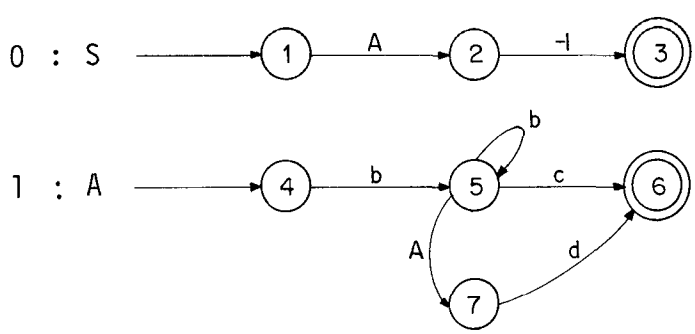
	AÇÃO				PROX	NUCLEO	FECHAMENTO
	a	b	c	⊣	A		
1	↓3				↓2	1	4
2	↓3		4	AC	↓5	2,7	4
3		4	6			5	
4			R(A)	R(A)		6	
5			4			7	
6	#3				#5	4	4

Note que PROX(1,A) = ↓2 . Empilha porque, apesar de $\delta_0(1,A) = 2$, $1 \in K_0$ e $4 \in K_1$. Como são autômatos diferentes não há problema.

No estado 6 percebe-se que a gramática G1 não é SLR(Ø,1): a função EMPILHA é igual a "#".

Este é o caso de recursão à esquerda com ciclo no estado inicial, no qual só posso saber se é fim de trecho reduzido olhando o símbolo anterior. No caso, c indica que não é fim.

EXEMPLO 2. Seja a gramática RRP G2:



	AÇÃO				PROX
	b	c	d	⊣	A
1	↓3				2
2			AC		
3	#5				4
4					

NUCLEO	FECHAMENTO
1	4
2	
5	4
7	

No estado 3 a função EMPILHA assume o valor "#" porque os estados 5 e 4 de seu NUCLEO e FECHAMENTO, respectivamente, pertencem ao mesmo autômato.

A é "self-embedding", logo a gramática G2 não é SLR(Ø,1). A linguagem gerada será:

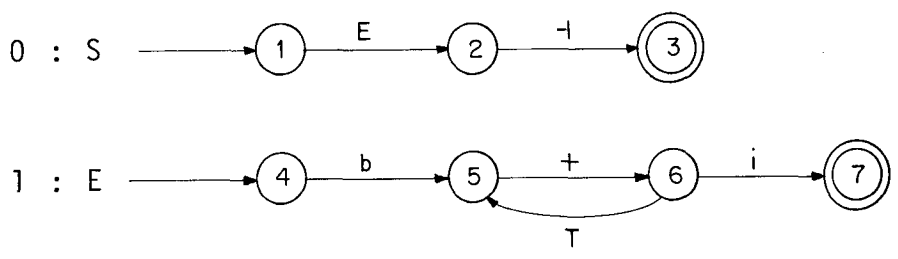
$$b^i bb^j cd^i$$

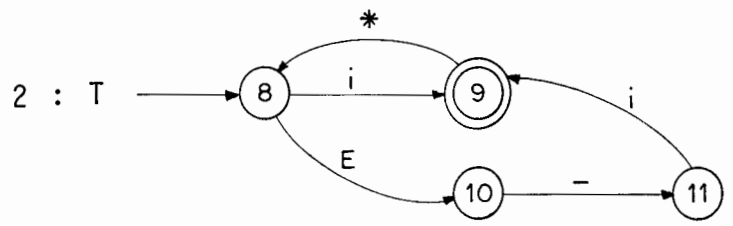
Como descobrir, no instante que chega c , quanto vale j? Sei apenas que entraram b^x e que

$$x = i + 1 + j$$

mas só vou conhecer o valor de i contando d^i , que ainda não chegaram.

EXEMPLO 3. Seja a gramática RRP G3:





	AÇÃO						PROX	
	⊣	i	+	b	*	-	E	T
1				↓3			2	
2	AC							
3			4					
4		#6		↓3			↓5	3
5					7			

NUCLEO	FECHAMENTO
1	4
2	
5	
6	8,4
10	

No estado 4 a função EMPILHA assume o valor "#" indicando que a gramática G3 não é SLR(Ø,1).

Analisando os estados do $NUCLEO_4$ e $FECHAMENTO_4$ vemos porque a função EMPILHA assume o valor "#" :

$\delta_E(6,i) = 7$ e

$\delta_T(8,i) = 9$

Por estes dois estados, como são de autômatos diferentes, poderíamos empilhar. Mas, como

$Q_E = 4$,

EMPILHA(4,i) = "#" , pois os estados 6 e 4 pertencem ao mesmo autômato.

3.6.3. O construtor e o analisador RRP SLR(Ø,1)

O construtor da tabela de controle pode ser definido agora sem maiores problemas.

ALGORITMO. Construtor SLR(Ø,1).

PASSO 1: Construa a máquina LERPARAFRENTE utilizando o algoritmo construtor MØ(G) e colocando as ações de redução e avanço conforme 3.6.1;

PASSO 2: Se a construção foi bem sucedida construa a função EMPILHA, conforme as regras (2), (3) e (4);

PASSO 3: Se EMPILHA(i,X) = "#" para algum par (i,X), assinale que a gramática não pertence à classe SLR(Ø,1).

A determinação do início de um trecho reduzido deveria necessitar,ainda, de uma segunda função que indicasse quando o desempilhamento deve parar, ou seja, quando $Q_j \in FECHAMENTO_t$ por ocasião de uma redução pela regra j . Mas essa informação já consta da tabela PROX, uma vez que

$$PROX(t,A_j) \neq erro \iff Q_j \in FECHAMENTO_t$$

Portanto, a máquina LERPARATRAS é constituída apenas pela função EMPILHA, e ocupa apenas um bit adicional por elemento da tabela. Para permitirmos o teste sobre PROX, substituiremos a ação "reduz j" por "reduz A_j". Esta substituição é meramente formal pois como cada categoria possui uma e uma só regra, a correspondência é biunívoca.

EXEMPLO. Tabela do Analisador RRP SLR(Ø,1) da gramática G da Fig.1.

Notação: ↓ empilha

↑ reduz

AC aceita

	AÇÃO							PROX		NUCLEO	FECHAMENTO
	-|	+	(	-	)	i	*	E	T		
1			↓4			↓5		2	↓3	1	4,6
2	AC									2	
3	↑E	6			↑E					5	
4			↓4	7		↓5		8	↓3	7	4,6
5	↑T	↑T			↑T		9			10	
6			↓4			↓5			3	4	6
7			↓4			↓5		8	↓3	8	4,6
8					5					9	
9			4			5				6	

Note que, no estado 4, a função EMPILHA não apresentou problemas porque os estados 7, 4 e 6 não têm transição em comum com nenhum símbolo.

Evidentemente estamos deixando de empilhar diversas tabelas que serão necessárias para a determinação da regra derivada e para o comando do tradutor. Em lugar de empilharmos essas tabelas e estabelecermos um sistema complicado de marca-

ção das tabelas na pilha para seleção das que indicam início de trecho reduzido, preferimos utilizar duas pilhas no analisador: a usual pilha sintática, onde entram as tabelas determinadas pela função EMPILHA, e uma pilha semântica - na falta de melhor termo - onde será empilhada alguma informação associada à transição: o número de uma rotina semântica, ou o símbolo utilizado ou um comando para o tradutor. As duas pilhas são associadas por ponteiros da pilha sintática para a pilha semântica. Na próxima seção discutiremos as vantagens dessa organização.

ALGORITMO. ANALISADOR RRP SLR(Ø,1).

comentários:

PSINT - pilha sintática, onde cada elemento da pilha é formado de um número de tabela e um ponteiro para a pilha semântica: PSINT(TABELA,PONTEIRO).

PSEM - pilha semântica.

SCAN - rotina que coloca o símbolo atual em ENTRADA

ESTADOATUAL- funciona como topo de PSINT.

TRATASEMANTICA - rotina que atua sobre a pilha semântica. Se considerarmos que empilharemos os símbolos de entrada na pilha semântica, poderemos ter os seguintes procedimentos nesta rotina:

TRATASEMANTICA(avança) - coloca o símbolo de entrada na pilha semântica;

TRATASEMANTICA(reduz) - tendo o ponteiro do ESTADOATUAL igual ao início do trecho reduzido na pilha semântica, retira o trecho reduzido para emitir e coloca na pilha semântica o lado esquer-

```
                                do da regra derivada.  Note que
                                se o trecho reduzido for  ε ,não
                                desempilha nada.

fim dos comentários;


begin
 ESTADOATUAL :=1 ;  (* estado inicial *)
 SCAN;
 loop
  case AÇÃO[ESTADOATUAL,ENTRADA] of
     AC : exit;
 avança n: begin
             ESTADOATUAL:=n;
             TRATASEMANTICA(avança);
             SCAN
            end;
 ↓avança n: begin
              PUSH.PSINT(ESTADOATUAL,topo(PSEM)); (* empilha *)
              ESTADOATUAL :=n;
              TRATASEMANTICA(avança);
              SCAN
             end;
  reduz A : begin
             while PROX[ESTADOATUAL,A] = erro
              do ESTADOATUAL := POP.PSINT ; (*desempilha o topo*)
             TRATASEMANTICA (reduz);
             if PROX[ESTADOATUAL,A] = ↓p
              then PUSH . PSINT(ESTADOATUAL,topo(PSEM));
             ESTADOATUAL:=PROX[ESTADOATUAL,A]
            end;
  end case;
 end loop;
end.                                                          ■
```

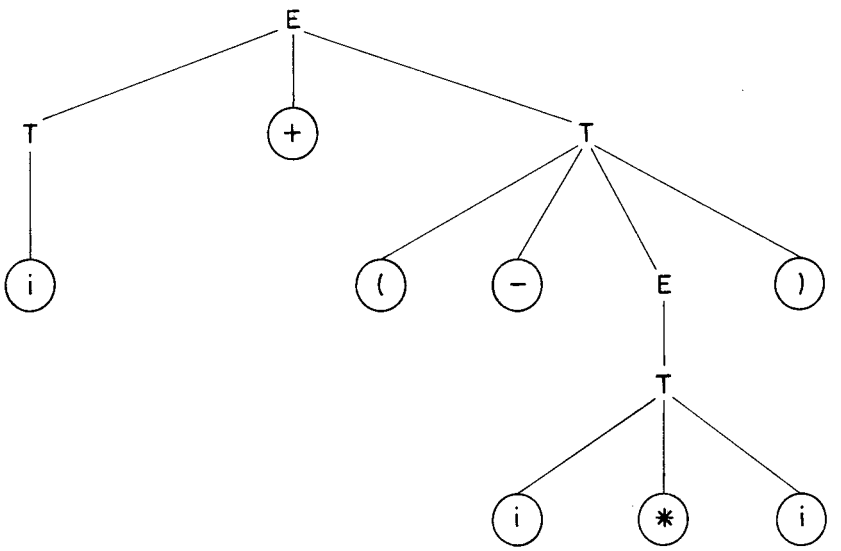
No exemplo a seguir, a pilha semântica conterá os próprios símbolos da transição e, por ocasião da redução, os emitirá como saída. O ponteiro da pilha sintática está indicado pela posição.

EXEMPLO 1. Análise sintática de (a+a)*a-| , tendo como base a tabela do exemplo anterior.

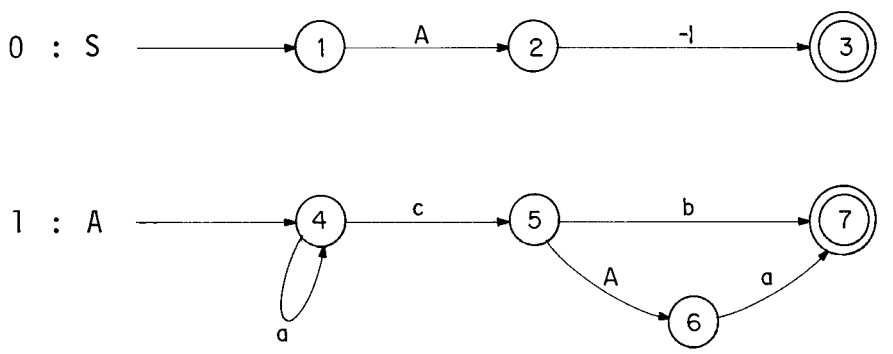
PILHAS SINTÁTICA / SEMÂNTICA	ESTADO ATUAL	REDUZIDO	ENTRADA	AÇÃO	SAÍDA
1	1		i+(-i*i)-|	↓5	
1 i	5		+(-i*i)-|	↑T	
1	1	T			i
1 T	3		+(-i*i)-|	6	
1 T +	6		(-i*i)-|	↓4	
1 6 T + (	4		-i*i)-|	7	
1 6 T + (-	7		i*i)-|	↓5	
1 6 7 T + (- i	5		*i)-|	9	
1 6 7 T + (- i *	9		i)-|	5	
1 6 7 T + (- i * i	5		)-|	↑T	
1 6 T + (-	7	T			i*i
1 6 7 T + (- T	3		)-|	↑E	
1 6 T + (-	7	E			T
1 6 T + (- E	8		)-|	5	

1 6	5	-|	↑T	
T + (- E)				
1	6	T		(-E)
T +				
1	3	-|	↑E	
T + T				
	1	E		T+T
	2	-|	AC	
E				

Árvore sintática produzida:



EXEMPLO 2. Seja a seguinte gramática RRP SLR(Ø,1):

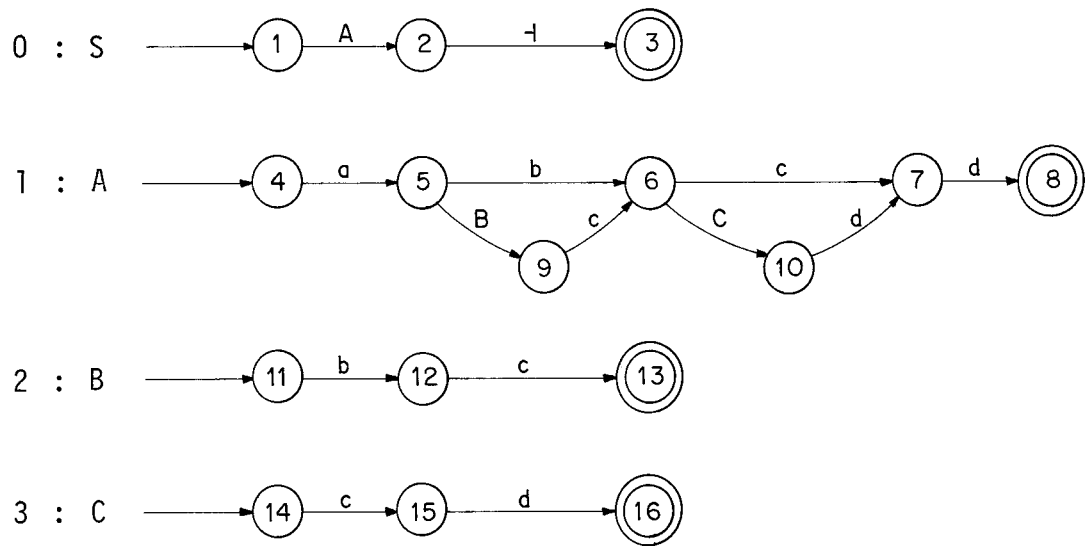


	AÇÃO				PROX	NUCLEO	FECHAMENTO
	a	b	c	-|	A		
1	↓3		↓4		2	1	4
2				AC		2	
3	3		4			4	
4	↓3	5	↓4		6	5	4
5	↑A			↑A		7	
6	5					6	

Análise sintática de aacacba-| :

PILHAS SINTÁTICA SEMÂNTICA	ESTADO ATUAL	REDUZIDO	ENTRADA	AÇÃO	SAÍDA
	1		aacacba-|	↓3	
1 a	3		acacba-|	3	
1 aa	3		cacba-|	4	
1 aac	4		acba-|	↓3	
1 4 aaca	3		cba-|	4	
1 4 aacac	4		ba-|	5	
1 4 aacacb	5		a-|	↑A	
1 aac	4	A			acb
1 aacA	6		a-|	5	
1 aacAa	5		-|	↑A	
	1	A			aacAa
A	2		-|	AC	

EXEMPLO 3. Seja a seguinte gramática RRP SLR(Ø,1):



	AÇÃO					PROX			NUCLEO	FECHAMENTO
	a	b	c	d	⊣	A	B	C		
1	↓3					2			1	4
2					AC				2	
3		↓4					5		5	11
4			↓6					7	6,12	14
5			8						9	
6			↑B	9					7,13,15	
7				10					10	
8			↓11					7	6	14
9				↑C	↑A				8,16	
10				12					7	
11				9					7,15	
12					↑A				8	

Análise sintática de abcccddd-| :

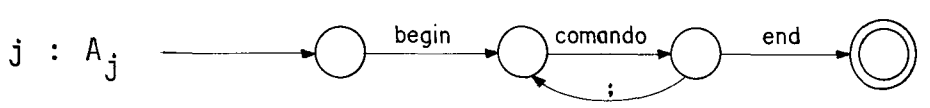
PILHAS SINTÁTICA / SEMÂNTICA	ESTADO ATUAL	REDUZIDO	ENTRADA	AÇÃO	SAÍDA
	1		abcccddd-|	↓3	
1	3		bcccddd-|	↓4	
a					
1 3	4		cccddd-|	↓6	
a b					
1 3 4	6		ccddd-|	↑B	
a b c					
1	3	B			bc
a					
1	5		ccddd-|	8	
a B					
1	8		cddd-|	↓11	
a B c					
1 8	11		ddd-|	9	
a B c c					
1 8	9		dd-|	↑C	
a B c c d					
1	8	C			cd
a B c					
1	7		dd-|	10	
a B c C					
1					
a B c C d	10		d-|	12	
1					
a B c C d d	12		-|	↑A	
	1	A		2	aBcCdd
A	2		-|	AC	

Análise sintática de abcd-| :

PILHAS SINTÁTICA / SEMÂNTICA	ESTADO ATUAL	REDUZIDO	ENTRADA	AÇÃO	SAÍDA
	1		abcd-|	↓3	
1	3		bcd-|	↓4	
a					
1 3	4		cd-|	↓6	
a b					
1 3 4	6		d	9	
a b c					
1 3 4	9		-|	↑A	
a b c d					
	1	A			abcd
	2		-|	AC	
A					

3.6.4. Controle do Empilhamento

O principal motivo de termos instituído duas pilhas no analisador é a possibilidade de controlarmos o empilhamento. As gramáticas RRP tendem a possuir regras sintáticas extremamente poderosas, capazes de gerar regras derivadas de comprimento ilimitado. Em análise ascendente normal, esse trecho permanece na pilha até o momento da redução. Isto leva a pilha a assumir um tamanho intolerável. Imaginemos que:



A pilha deverá conter um elemento para cada "comando" no interior do bloco. |LaLonde,abril/81| propõe a criação de um analisador capaz de tratar um atributo adicional (empilha ou não empilha) para contornar esse problema. Entretanto, seu método transfere esse atributo para as tabelas LR, uma vez que a pilha é única. É fácil de prever a complexidade de seu analisador pois, quando ao desempilhar para uma determinada redução, pode não encontrar a tabela que contém as informações para reiniciar o avanço, porque esta simplemente não foi empilhada. Além dessa complexidade, seu método ainda reduz a classe de gramáticas pois o atributo "não empilha" não pode ser distribuído arbitrariamente pela gramática (como, por exemplo, em todas as transições).

Nosso método separa as informações sintáticas das informações semânticas (conforme, aliás, recomendação em |LaLonde, janeiro/81|). A pilha sintática contém apenas o indispensável e a pilha semântica é absolutamente livre. Note, também, que esta segunda pilha a rigor existe em qualquer analisador ascendente e apenas a associamos ao analisador sintático.

3.7. Conclusões

As gramáticas com lados direitos regulares (RRP) ou gramáticas livres de contexto estendidas (ECF) são hoje, pelo menos, a forma predileta de apresentar aos usuários a sintaxe das linguagens de programação. Entretanto, os analisadores ascendentes, construídos diretamente a partir de gramáticas RRP,

ainda são suficientemente complexos para que os geradores de compiladores mais recentes prefiram se utilizar de gramáticas BNF, mesmo que geradas automaticamente a partir de gramáticas RRP (|DeRemer,80| e |DeRemer,81|). A principal dificuldade é a determinação do início de um trecho reduzido (handle), uma vez que as regras das gramáticas RRP podem dar origem a formas sentenciais de comprimento variável.

Esta tese traz, fundamentalmente, três novas contribuições ao problema de análise sintática ascendente para gramáticas RRP. Em primeiro lugar, define condições suficientes para determinar se uma gramática é RRP SLR(Ø,1), partindo da hipótese de que os atuais analisadores RRP LR são complicados porque são mais poderosos do que o necessário. Em segundo lugar, apresenta um analisador RRP SLR(Ø,1) que é compacto, fácil de construir e de implementar. Finalmente, introduz a idéia de se utilizar pilhas paralelas para análise sintática e comando de tradução e semântica, permitindo liberdade de empilhamento para ambos os casos. Esta solução evita os atuais métodos, bastante complexos |LaLonde,abril/81|.

Acreditamos ter proposto um método de análise sintática que se compara favoravelmente aos seus similares atuais. Sua eficiência deriva de alguma restrição sobre a gramática,que entretanto não afeta grandemente sua potência, e de uma gerência da pilha sintática durante a fase de avanço.

|Celentano,81| desenvolve outro processo de controle da pilha durante a fase de avanço, para gramáticas ECF. Seu método, entretanto, além de criar estados adicionais, pretende o controle integral da pilha e acaba por restringir enormemente

a classe de gramáticas abrangidas.

O analisador aqui proposto é extremamente simples e compacto, pois sua máquina de LERPARAFRENTE é a usual e a máquina de LERPARATRAS ocupa apenas um bit de espaço em cada ponto da tabela. Isto nos permite dizer que as tabelas de controle do analisador são da mesma ordem de grandeza - e normalmente menores - que as das gramáticas BNF equivalentes. Além disso, a complexidade de tempo do analisador é idêntica a do analisador BNF correspondente.

O método proposto aceita as classes de gramáticas contidas na classe RRP SLR(Ø,1): gramáticas RRP LR(Ø,Ø), gramáticas BNF SLR(1) e gramáticas BNF LR(Ø). Em particular, nos dois últimos casos, produz saída equivalente a do gerador usual dessas classes de gramáticas BNF.

A extensão deste método para gramáticas RRP LR(Ø,K) e derivadas é trivial, como pode ser visto em |Simone,82| que apresentou o analisador RRP LR(Ø,1).

BIBLIOGRAFIA

1. AHO, A.V. e ULLMAN, J.D., "The Theory of parsing, translation and compiling", vol.1, Prentice-Hall, 1972.

2. AHO, A.V. e ULLMAN, J.D., "Principles of Compiler Design", Addison-Wesley, 1977.

3. HOPCROFT, J.E. e ULLMAN, J.D., "Introduction to Automata Theory, Languages, and Computation", Addison-Wesley, 1979.

4. CELENTANO, A., "An LR parsing technique for extended context-free grammars", Computer Languages, vol.6, pp.95-107,1981.

5. DeREMER, F.R., "Extended LR(K) grammars and their parsers", Univ. California, Sta. Cruz, 1970.

6. DeREMER, F.R., PENELLO, T. e MEYERS, R., "A Syntax diagram for (preliminary) ADA", SIGPLAN NOTICES, vol.15, nº 7 e 8, julho-agosto, 1980.

7. DeREMER, F.R., PENELLO, T. e McKEEMAN, W.M., "Ada Syntax chart", SIGPLAN NOTICES , vol.16, nº 9, pp.48-59, setembro 1981.

8. DE SIMONE, E. e PEREIRA, L.C., "Algoritmos para gramáticas RRP SLR(1)", Anais do VIIº SEMISH, Soc.Bras.Computação, Campinas, julho 1980.

9. LaLONDE, W.F., "Practical LR analysis of regular right part grammars", Tese Ph.D., Dep.Comp.Science, Univ. Waterloo, outubro 1975.

10. LaLONDE, W.F., "Constructing LR parsers for regular right part grammars", Acta Informática, 11, pp.177-193, 1979.

11. LaLONDE, W.F., e DES RIVIERES, J., "Handling operator precedence in arithmetic expressions with tree transformations", TOPLAS, 3, nº 1, janeiro 1981.

12. LaLONDE, W.F., "The construction of stack-controlling LR parsers for regular right part grammars", TOPLAS, 3, nº 2, abril 1981.

13. MADSEN, D.L., e KRISTENSEN, B.B., "LR parsing of extended context-free grammars", Acta Informatica, 7, pp.61-73, 1976.

14. TELLES, A.A.S. e DE SIMONE, E., "Gerador de analisadores sintáticos RRP LL(1)", Anais VIII SEMISH, Soc.Bras.Comp., Florianópolis, julho 1981.

15. De SIMONE, E. e PEREIRA, L.C., "Análise sintática RRP LR(Ø,1)", Revista Brasileira de Computação, nº 3, vol.1, fev.82.